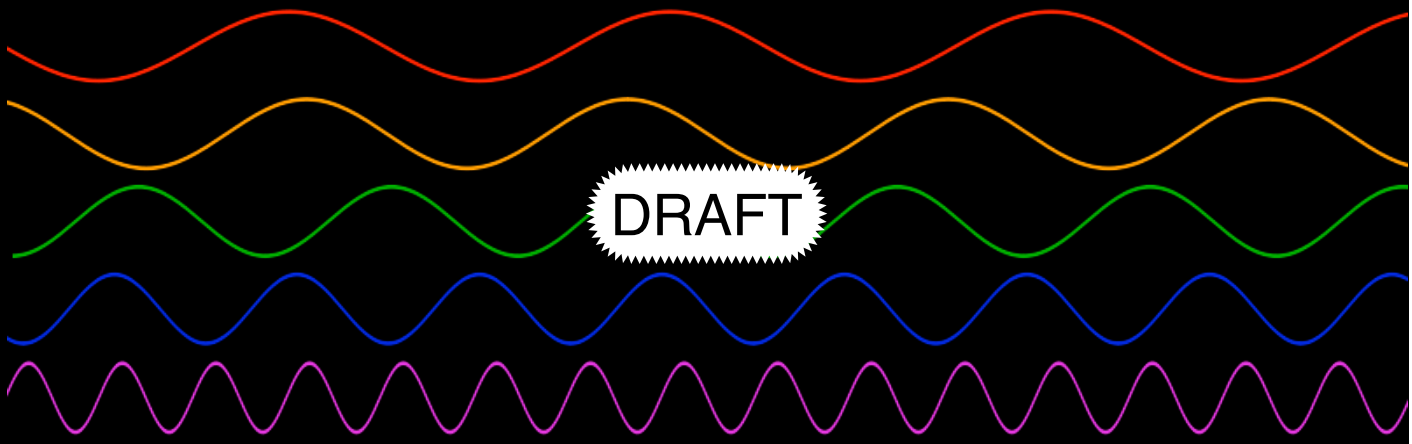
Statistical analysis of flow data using Python and Redis



FLOCON 2013

Kevin Noble
Terraplex@gmail.com

Overview

- 1. Beacon description
- 2. Beacons as used by attackers
- 3. Considerations for beacon classification
  - a. periodicity in time series analysis
    - i. Considerations to evaluate periodicity
- 4. Visualize beacons
  - a. Factors of classification useful to detect beacons
- 5. Beacon Bits, an analytical tool set and workflow to detect beacons
  - a. Demo
  - b. Extracting data from flows
  - c. Storing timing data
  - d. Statistical analysis and evaluation of beacon properties
- 6. Result
- 7. Code / Discussion / Q&A

http://www.mcafee.com/us/resources/white-papers/wp-global-energy-cyberattacks-night-dragon.pdf

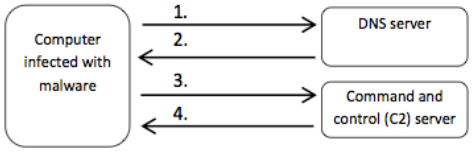
Network Communications

Network communications are relatively easy to detect because the malware uses a unique host beacon and server response protocol. Each communication packet between the compromised host and the C&C server is signed with a plain text signature of "hW$." (or "\x68\x57\x24\x13") at the byte offset 0x42 within the TCP packet.

The backdoor begins its beacon at approximately five-second intervals with an initial packet that may be detected with the pattern: "\x01\x50[\x00-\xff]+\x68\x57\x24\x13."

http://www.commandfive.com/papers/C5_APT_C2InTheFifthDomain.pdf

The malware used in the attack was programmed to communicate with several 'callback' domains. The malware located its C2 server(s) by resolving these domains into IP addresses using the ubiquitous Domain Name System (DNS) [3] protocol. These communications are depicted in Figure 1.



1. Using the Domain Name System (DNS) protocol, the computer asks a DNS server for directions to the callback domain.
2. The DNS server advises that the callback domain is located at IP address x.x.x.x.
3. The malware communicates with the C2 server located at IP address x.x.x.x to obtain C2 instructions and/or to send a response.
4. The C2 server provides additional C2 instructions to the malware.

After sending the basic beacon, the compromised computers waited for a response from the server, then closed the connection when they had not received a response from the server within five seconds.

Both of the compromised computers reattempted the communications approximately every eight seconds. On some days the high frequency of the beacon activity resulted in over 10000 connection attempts per victim in a 24 hour period.

Beacons

1. Beacons manifest as repetitious communication attempts in the form of packets
   - a. Most beacons are not malicious
   - b. Malicious beacons are sourced from infected host where the malware repeatedly attempts remote connectivity
   - c. Beacon events are discernible
2. Detection
   - a. The more frequent a beacon, the easier to detect
   - b. Beacons that are consistent in time series are easier to detect
   - c. Beacons events lend themselves to time series analysis

Timing is a signature

PROTOCOL	TYPICAL BEACON INTERVAL* (SECONDS)
LURK	26
X-Shell C601	36
Update?	1 to 13, 12±3, 16, 104±3 or 200 ±15
Murcy	11
Oscar	12±2, 13, 15, 16, (55 or 155±5), (7.5, 8.5 or 15) , (45, 55, 106)
BB	8
DB	4 to 92
Qdigit	60

* Commas indicate that the interval changed between victims. Brackets indicate that a variety of intervals were observed from a single computer.

TABLE 7: INTERVAL BETWEEN COMMUNICATIONS

http://www.commandfive.com/papers/C5_APT_C2InTheFifthDomain.pdf

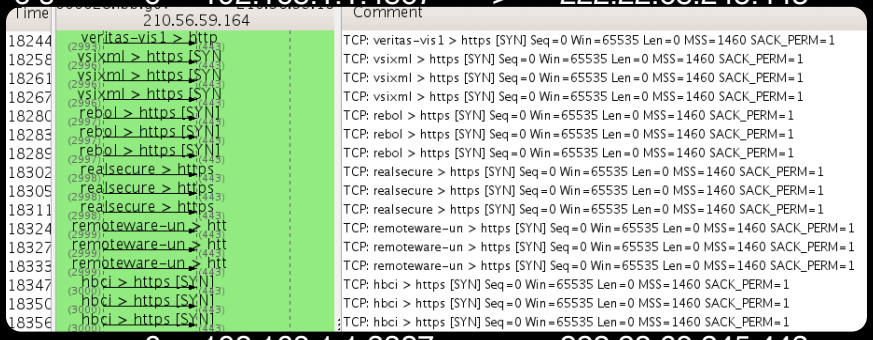
Sample Beacon as viewed in flow for network and timing properties

Present all the characteristics and properties for known beacons

Avoid payload analysis (except perhaps size)

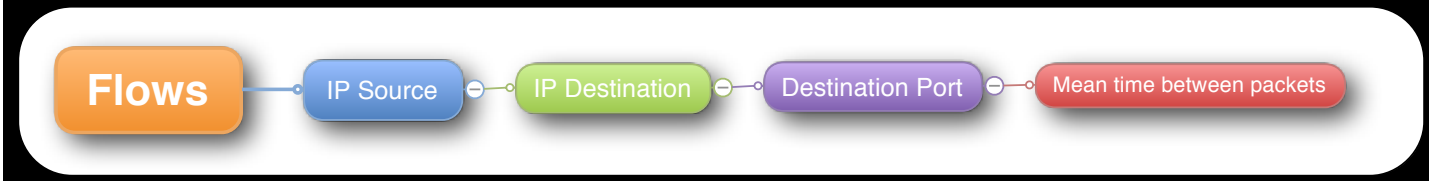
beacon/testset$ ra -nnr beacon_test_extract.arg - host 222.22.68.245

StartTime	Flgs	Proto	SrcAddr	Sport	Dir	DstAddr	Dport	TotPkts	TotBytes	State
13:00:58.783986	e s	6	192.168.1.1.3719		->	222.22.68.245.443		2	124	REQ
13:31:52.667327	e s	6	192.168.1.1.3208		->	222.22.68.245.443		2	124	REQ
14:01:53.659479	e s	6	192.168.1.1.2665		->	222.22.68.245.443		2	124	REQ
14:32:00.062273	e s	6	192.168.1.1.2152		->	222.22.68.245.443		2	124	REQ
15:02:55.611042	e s	6	192.168.1.1.1962		->	222.22.68.245.443		2	124	REQ
15:33:52.663009	e s	6	192.168.1.1.1524		->	222.22.68.245.443		2	124	REQ
16:03:52.602414	e s	6	192.168.1.1.4867		->	222.22.68.245.443		2	124	REQ
16:33:57.090316	e s	6	192.168.1.1.3327		->	222.22.68.245.443		2	124	REQ
17:04:52.558100	e s	6	192.168.1.1.2808		->	222.22.68.245.443		2	124	REQ
17:34:59.598407	e s	6	192.168.1.1.2231		->	222.22.68.245.443		2	124	REQ
18:05:56.669750	e s	6	192.168.1.1.1718		->	222.22.68.245.443		2	124	REQ
18:36:53.968150	e s	6	192.168.1.1.1244		->	222.22.68.245.443		2	124	REQ
19:06:56.229070	e s	6	192.168.1.1.4999		->	222.22.68.245.443		2	124	REQ
19:37:53.975195	e s	6	192.168.1.1.4680		->	222.22.68.245.443		2	124	REQ
20:08:53.685264	e s	6	192.168.1.1.4361		->	222.22.68.245.443		2	124	REQ
20:38:54.173905	e s	6	192.168.1.1.4042		->	222.22.68.245.443		2	124	REQ
21:10:09.140943	e s	6	192.168.1.1.3723		->	222.22.68.245.443		2	124	REQ
21:40:52.834383	e s	6	192.168.1.1.3404		->	222.22.68.245.443		2	124	REQ
22:10:57.850103	e s	6	192.168.1.1.3085		->	222.22.68.245.443		2	124	REQ
22:41:55.148182	e s	6	192.168.1.1.2766		->	222.22.68.245.443		2	124	REQ
23:12:58.582524	e s	6	192.168.1.1.2447		->	222.22.68.245.443		2	124	REQ
23:43:52.478378	e s	6	192.168.1.1.4999		->	222.22.68.245.443		2	124	REQ



GOAL: Surface malicious beacons for inspection by examining Network traffic

Inspecting traffic flows for beacons
Flow based tools have a limited facility to detect beacons alone.
Flow tools are ideal for the collection and verification of beacons.
Flow based tools do provide counts and summaries and quantizing (bins) in some cases.
Quantize time to seconds (sub-seconds complicate the details) appears to be useful.
Timing is the key to detection followed by verification by inspecting the host.

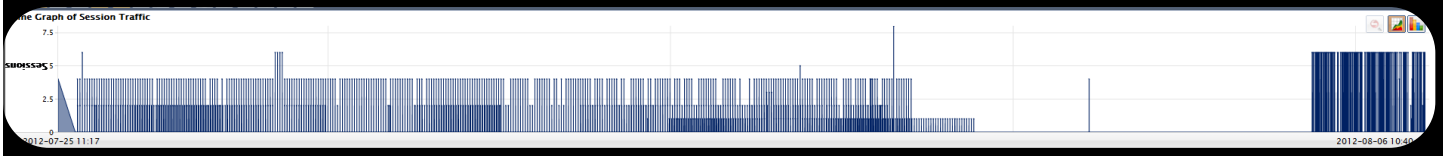


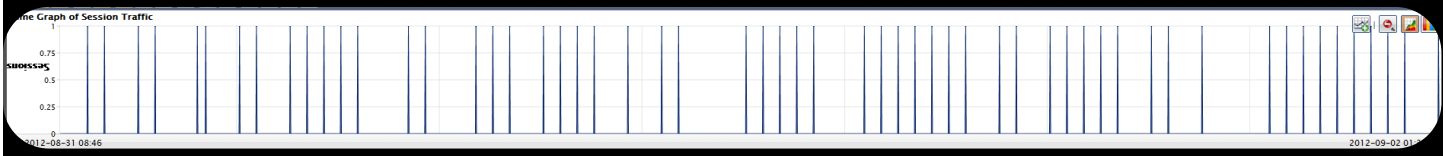
Beacon p0rn

Visual timing as a graph

Produces an instant visual representation of a beacon.

Graphing does not scale to allow analyst to inspect everything.





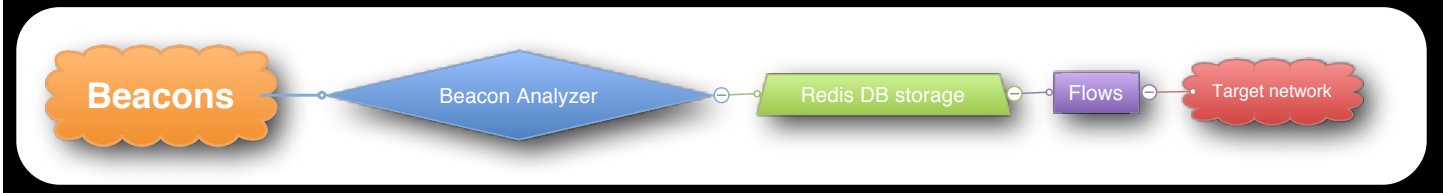
[1854, 1801, 1807, 1855, 1857, 1800, 1805, 1855, 1807, 1857, 1857, 1803, 1857, 1860, 1801, 1843, 1805, 1858, 1863, 1854, 1801, 1863, 1859, 1857, 1801, 1859, 1802, 1858, 1802, 1802, 1856, 1800, 1800, 1800, 1860, 1804, 1858, 1863, 1859, 1857, 1804, 1802, 1854, 1804, 1856, 1802, 1859, 1812, 1847, 1808, 1853, 1867, 1851, 1800, 1800, 1806, 1801, 1854, 1801, 1800, 1865, 1861, 1861, 1850, 1800, 1800, 1801, 1864, 1858, 1857, 1803, 1804, 1853, 1801, 1864, 1859, 1802, 1859, 1858, 1857, 1803, 1808, 1849, 1804, 1857, 1800, 1808, 1853, 1863, 1861, 1854, 1802, 1858, 1865, 1857, 1865, 1855, 1802, 1856, 1800, 1803, 1862, 1859, 1858, 1801, 1800, 1859, 1806, 1853, 1859, 1801, 1804, 1801, 1855, 1812, 1803, 1844, 1800, 1802, 1858]

Graphing every session does not scale

Beacon detection

Beacon Bits

- 1. Parse from FLOW
  - a. IP Source
  - b. IP Dest
  - c. Port Dest
  - d. Time (from Source)
- 2. DataStore
  - a. Native Python
  - b. Redis
- 3. Analysis
  - a. Python
- 4. BEACONS



```
IP source   1.1.1.1
IP dest     210.215.10.254  "NEXONASIAPACIFIC"
dst port    443
pair_count  8432
mean        121
Standard Deviation: 0.026849474628 169643.0
compensated_variance: 2542
online_variance: 20548
online_variance_n: 20546
web_std_dev (0.002493930934161027, 0.22931978029843433)
seconds     1020272     minutes     17004 hours 283
    days    11
src_count   10809
dst_count   8432
traffic with source and dest:
'SET:1.1.1.1:210.215.10.254:443:2012810'
'SET:1.1.1.1:210.215.10.254:443:2012811'
'SET:1.1.1.1:210.215.10.254:443:2012812'
'SET:1.1.1.1:210.215.10.254:443:2012813'
'SET:1.1.1.1:210.215.10.254:443:2012814'
'SET:1.1.1.1:210.215.10.254:443:2012815'
'SET:1.1.1.1:210.215.10.254:443:2012816'
'SET:1.1.1.1:210.215.10.254:443:2012817'
'SET:1.1.1.1:210.215.10.254:443:2012818'
'SET:1.1.1.1:210.215.10.254:443:2012819'
'SET:1.1.1.1:210.215.10.254:443:2012820'
'SET:1.1.1.1:210.215.10.254:443:2012821'
'SET:1.1.1.1:210.215.10.254:443:2012822'
'SET:1.1.1.1:210.215.10.254:443:multi']
[21, 223, 21, 223, 21, 222, 21, 223, 21, 223, 21, 223, 21, 222, 21, ….]
```

Beacon Classification and expression

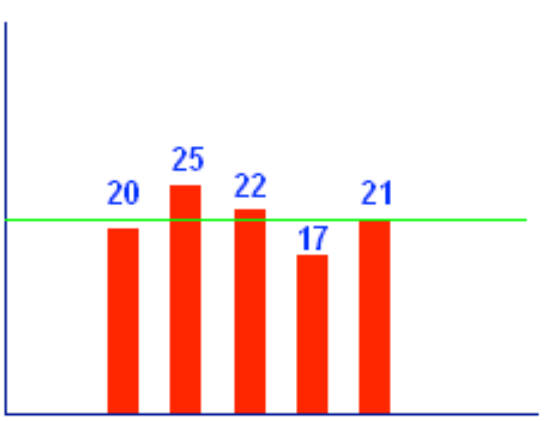
Execution condition	Frequency	Interval / Mean	Packet Proto	Packet Dest	Port	Payload	Payload Size
Continuous	Consistent	Static	Single	Single	Single	Consistent	Static
conditional	Transient	Dynamic	Multiple	Multiple	Multiple	Transient	Dynamic
transient						none	

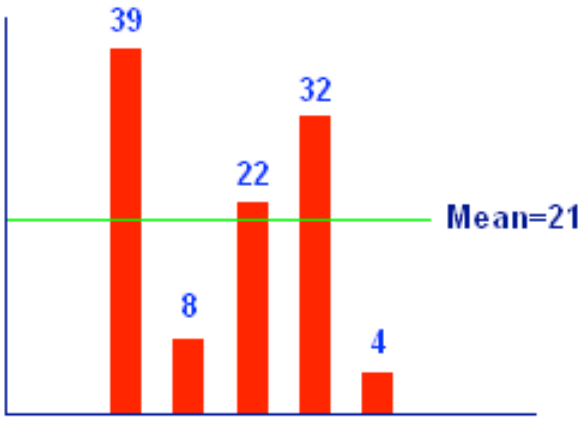
Beacon expression as a combination of conditions

Continuous and consistent TCP packets at 300 second intervals
TCP packet over a single port 80 every 900 seconds continuously
7 packets, 5 minutes apart, every 3 days using TCP or UDP to one of of 5 host over one of these 3 ports, with the following payload
1 TCP packet, every 30 day to one of 30 possible host

Malicious Beacons top characteristics used in the analysis process

- 1. Unconnected beacons
  - a. Low Varience
  - b. Low Standard Deviation
  - c. Limited number of host attempting to Connect
  - d. At least 3 packets
  - e. At least 15 minutes of 'total' time in the analysis
- 2. Connected beacons
  - a. Similar as unconnected
  - b. Payload is a factor
    - i. Strings / offsets / atomic





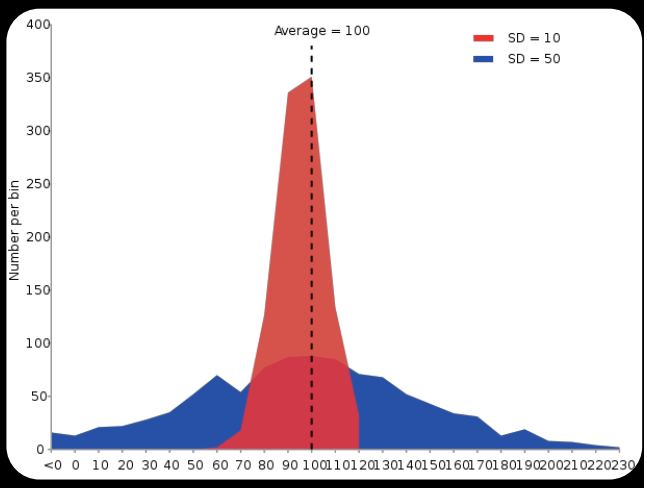
Histograms

- 1. Limited usefulness if used exclusively
- 2. Histograms value factors:
  - a. Large sample population
  - b. Combined with varience
  - c. Combined with static classifications (previous slides)
- 3. Dropped from analysis based on performance of other factors

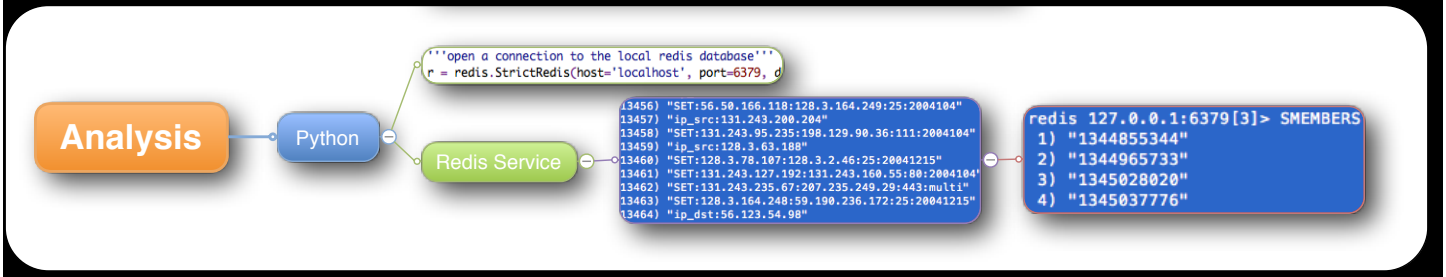
```
$ rahisto -H dur 120 -r htest.arg
N =         122
mean =      2.927606
stddev =    0.382476
max =       3.113874
min = 0
median
95% = 3
mode = 0.000000
```

Flow conversion to mysql
rasqltimeindex -r argus.file -w mysql://user@host/db

Class	Interval	Freq	Rel.Freq	Cum.Freq
108	2.782000e+00	0	0.0000%	1.6393%
109	2.808000e+00	1	0.8197%	2.4590%
110	2.834000e+00	2	1.6393%	4.0984%
111	2.860000e+00	3	2.4590%	6.5574%
112	2.886000e+00	8	6.5574%	13.1148%
113	2.912000e+00	16	13.1148%	26.2295%
114	2.938000e+00	25	20.4918%	46.7213%
115	2.964000e+00	19	15.5738%	62.2951%
116	2.990000e+00	20	16.3934%	78.6885%
117	3.016000e+00	10	8.1967%	86.8852%
118	3.042000e+00	5	4.0984%	90.9836%
119	3.068000e+00	8	6.5574%	97.5410%
120	3.094000e+00	3	2.4590%	100.0000%



Enumerate over keys
Should be able to move through the millions of keys quickly
Evaluate traffic based on timing properties in a statistical sense
Some assumption include host might be up during working hours
No more then 4 host would be infected




```
sets_sub = each.split(':')
set_src_ip = sets_sub[1]
set_dst_ip = sets_sub[2]
set_dst_port = sets_sub[3]
set_date = sets_sub[4]
src_count = r.get('ip_src:'+set_src_ip)
dst_count = r.get('ip_dst:'+set_dst_ip)
```

Variance

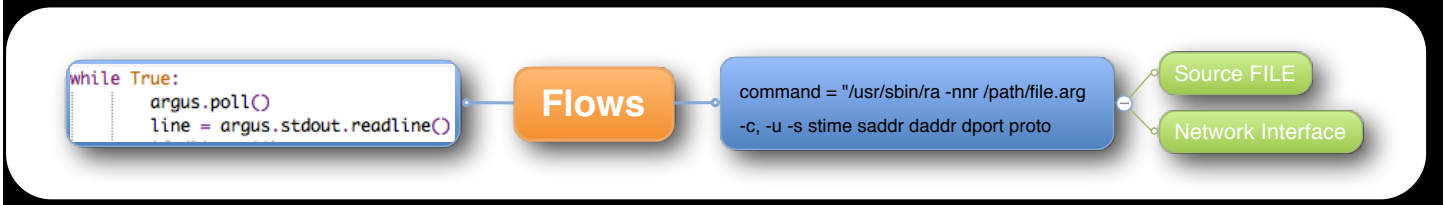
1. http://en.wikipedia.org/wiki/Algorithms_for_calculating_variance
   - a. **Algorithms for calculating variance** play a major role in statistical computing. A key problem in the design of good algorithms for this problem is that formulas for the variance may involve sums of squares, which can lead to numerical instability as well as to arithmetic overflow when dealing with large values.
   - b. Several Algorithms tested, settled on using three:
     - i. Compensated Variance
     - ii. Online variance
     - iii. Kurtosis

Standard Deviation

- 1. Little ‘dispersion’ for each set
- 2. Minimum population distance from the mean
- ▼ 3. Using a MODIFIED version of Standard Deviation that would be considered a WEIGHT
  - a. Tolerance increase with frequency (reverting to normal standard deviation for final release)

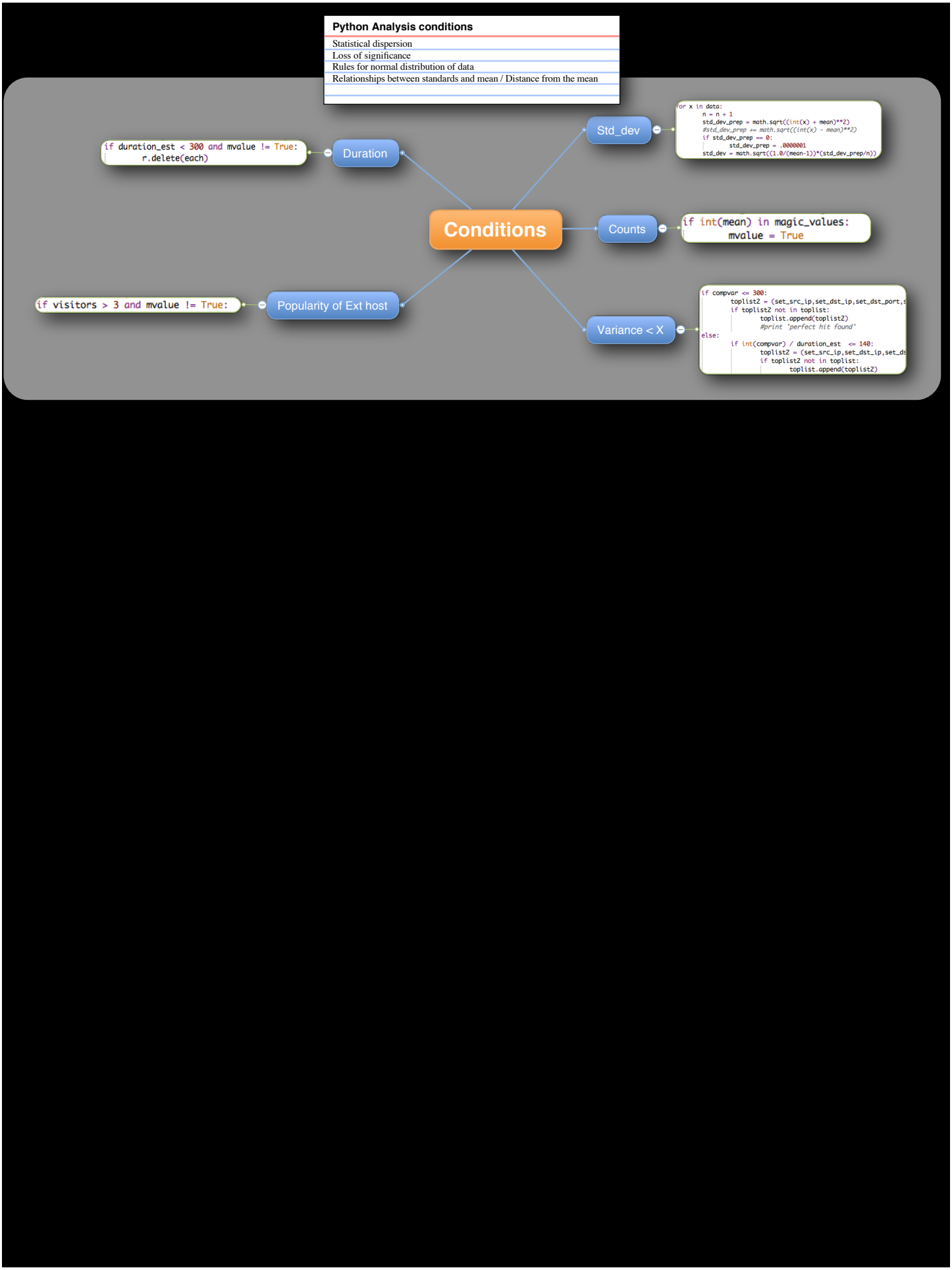
SOURCE IP	DEST IP	DEST PORT	DATE	STDDEV
100.0.5.230	1.0.20.5	8888	2012913	0.045732737
100.0.5.230	1.0.20.5	8888	2012914	0.044662676
100.0.5.230	1.0.20.5	8888	2012915	0.04343173
100.0.5.230	1.0.20.5	8888	2012916	0.042813404
100.0.5.230	1.0.20.5	8888	multi	0.019851071

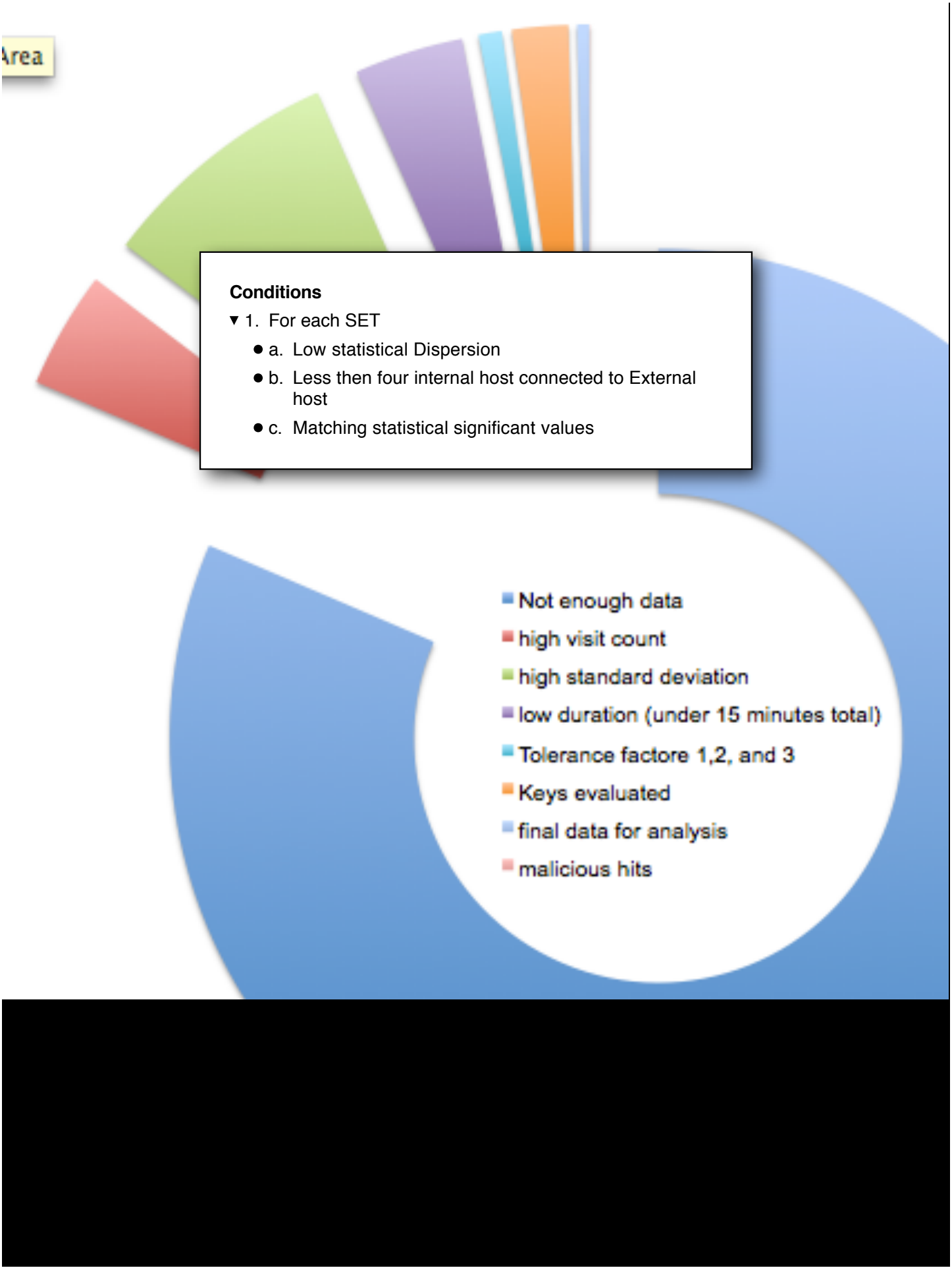
Can we tabulate timing for traffic as a means to detect beacons?
TCP SYN
Isolated to traffic sources from the network we seek to defend
Traffic destined to external network (avoid internal to internal packets)
Exclusion of trusted and authorized host and networks (if possible)
Limited totTrack timing properties



Polling
Using Python to compile a dataset is a process of conversion from binary parsed to text, formed into sets.
The largest sample set took 54 minutes to consume and held traffic for 16 days.
Python handles the sets fairly well but does not facilitate continuous analysis.

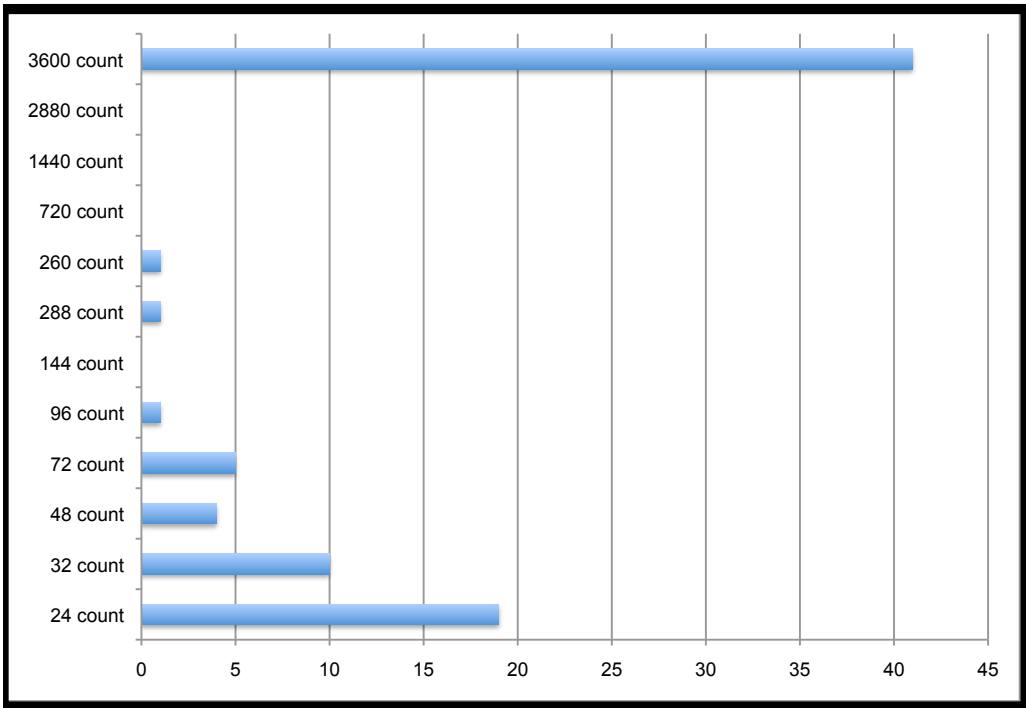
Analysis considerations




Area
Conditions
1. For each SET
a. Low statistical Dispersion
b. Less then four internal host connected to External host
c. Matching statistical significant values
Not enough data
high visit count
high standard deviation
low duration (under 15 minutes total)
Tolerance factore 1,2, and 3
Keys evaluated
final data for analysis
malicious hits

Divisible by 60 seconds?
Beacons generally resolve to set intervals in minutes
Connected sessions also maintain a connected state set in minutes
Most basic Remote Administration Tools
False positive are frequent
Evaluating Interval count alone still produces a useful set
Excluding trusted networks is useful

seconds in a day	Interval in minutes	Count
86400	0.5	2880
86400	1	1440
86400	2	720
86400	4	360
86400	5	288
86400	10	144
86400	15	96
86400	20	72
86400	30	48
86400	45	32
86400	60	24



Interval	Count
0.5	30
1	60
2	120
4	240
5	300
10	600
15	900
20	1200
30	1800
45	2700
60	**3600**
40	2400
30	1800
20	1200

Introduction to Redis

Redis is an open source, advanced **key-value store**. It is often referred to as a **data structure server** since keys can contain strings, hashes, lists, sets and sorted sets.

You can run **atomic operations** on these types, like appending to a string; incrementing the value in a hash; pushing to a list; computing set intersection, union and difference; or getting the member with highest ranking in a sorted set.

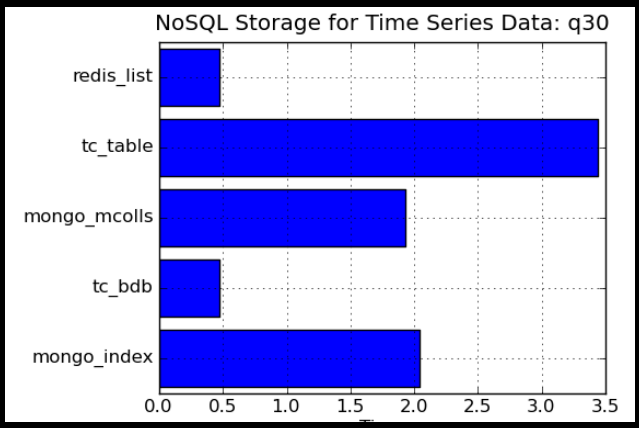
In order to achieve its outstanding performance, Redis works with an **in-memory dataset**. Depending on your use case, you can persist it either by dumping the dataset to disk every once in a while, or by appending each command to a log.

Redis also supports trivial-to-setup master-slave replication, with very fast non-blocking first synchronization, auto-reconnection on net split and so forth.

Other features include a simple check-and-set mechanism, pub/sub and configuration settings to make Redis behave like a cache.

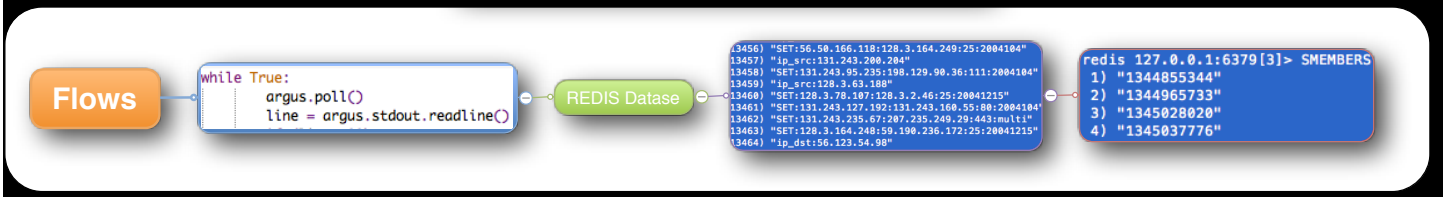
You can use Redis from most programming languages out there.

Redis is written in **ANSI C** and works in most POSIX systems like Linux, *BSD, OS X without external dependencies. Linux and OSX are the two operating systems where Redis is developed and more tested, and we **recommend using Linux for deploying**. Redis may work in Solaris-derived systems like SmartOS, but the support is *best effort*. There is no official support for Windows builds, although you may have some options.



Source: https://github.com/yinhm/nosql-tsd-benchmark

Collection
Tracking SETS with timing information
Tracking Source IP activity by count
Tracking Destination activity by count
Redis manages duplicates
Redis can handle the size
Memory is ideal for the transaction rate and the type of data being managed




```
beacon/testset$ ra -nnr beacon_test_extract.arg  - host 222.22.68.245
      StartTime  Flgs  Proto       SrcAddr  Sport  Dir       DstAddr  Dport  TotPkts  TotBytes  State
 13:00:58.783986  e s      6  192.168.1.1.3719   ->  222.22.68.245.443      2       124  REQ
 13:31:52.667327  e s      6  192.168.1.1.3208   ->  222.22.68.245.443      2       124  REQ
 14:01:53.659479  e s      6  192.168.1.1.2665   ->  222.22.68.245.443      2       124  REQ
 14:32:00.062273  e s      6  192.168.1.1.2152   ->  222.22.68.245.443      2       124  REQ
 15:02:55.611042  e s      6  192.168.1.1.1962   ->  222.22.68.245.443      2       124  REQ
```

Simplistic data schema

1. For Each IP Source, IP Dest, Dest Port, Date
    - a. Unix Time (String)
2. Counts
    - a. Increment counter
        - i. Source
        - ii. Destination
3. Date and Multiple
    - a. Supports differential analytical output
    - b. Statistical significance might be represented over multiple days
    - c. Statistical significance might be represented on a single day
4. Expiring keys
    - a. Necessary for production
5. White List
    - a. Useful for production
    - b. Requires care and feeding

Demonstration

- 1. start redis server and client
- 2. Populate redis database from flow file
- 3. collect timing data form flow file
- 4. launch analyzer
  - a. show redis db post analyzer
- 5. launch graph view

Significance

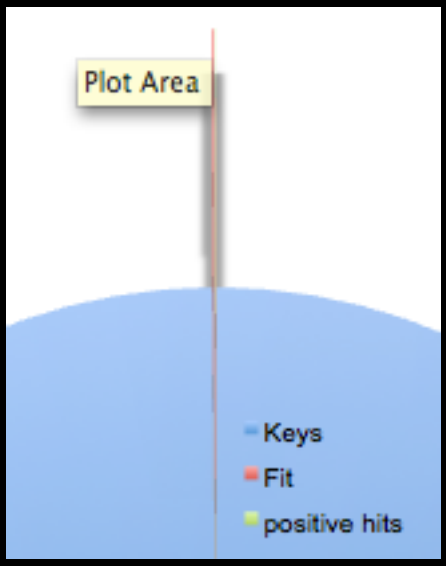
Parsing through 3 days of traffic yields beacons.

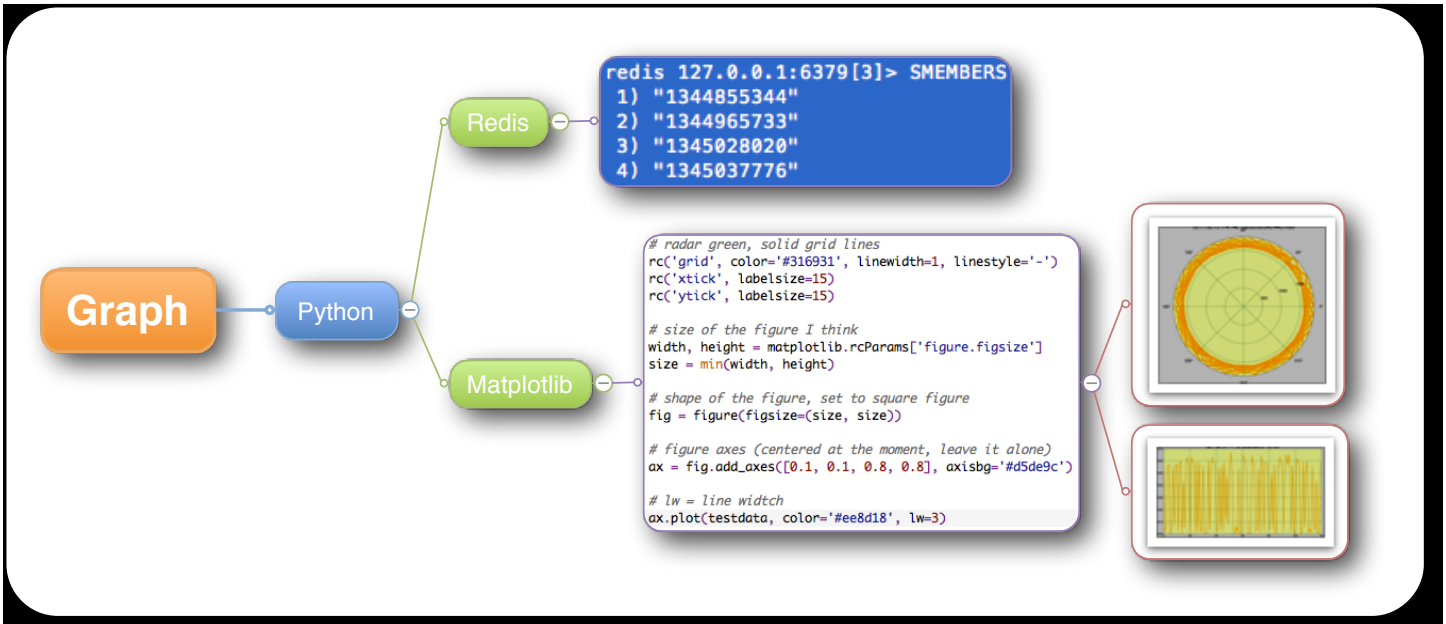
The number of beacons depends on the test conditions

The most statistically significant data included malicious beacons

Pulling the most significant results with flows and full packet capture is useful

Host inspection is the best verification of results




Graph
Python
Redis
redis 127.0.0.1:6379[3]> SMEMBERS
1) "1344855344"
2) "1344965733"
3) "1345028020"
4) "1345037776"
Matplotlib
# radar green, solid grid lines
rc('grid', color='#316931', linewidth=1, linestyle='-')
rc('xtick', labelsize=15)
rc('ytick', labelsize=15)
# size of the figure I think
width, height = matplotlib.rcParams['figure.figsize']
size = min(width, height)
# shape of the figure, set to square figure
fig = figure(figsize=(size, size))
# figure axes (centered at the moment, leave it alone)
ax = fig.add_axes([0.1, 0.1, 0.8, 0.8], axisbg='#d5de9c')
# lw = line width
ax.plot(testdata, color='#ee8d18', lw=3)

Graph / Plot (text view)

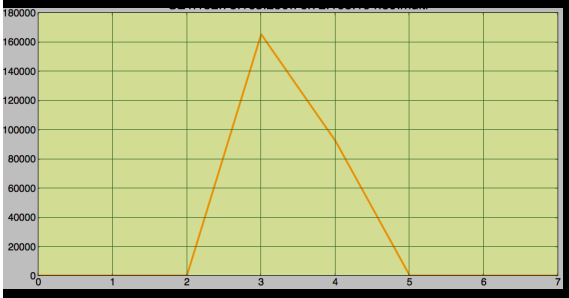
1. Specific results can be examined in detail
2. The timing data can be put into an array for a graphical display

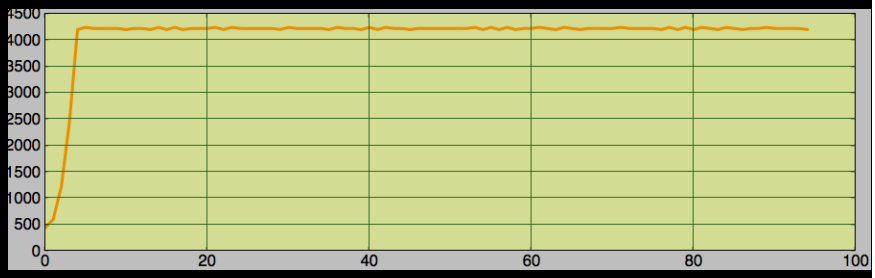
```
seconds between query: 21 Standard Deviation: 0.0229724128309 425812.0
IP source       100.0.9.95
IP dest         1.0.9.25        "APNIC Debogon Project"
dst port        80
pair_count      1896
mean            224
seconds         424704  minutes         7078 hours 117  days    4
src_count       1899
dst_count       3792
traffic with source and dest ['SET:100.0.9.95:1.0.9.25:80:2012912', 'SET:100.0.9.95:1.0.9.25:80:2012913', 'SET:100.0.9.95:1.0.9.25:80:2012914',
'SET:100.0.9.95:1.0.9.25:80:2012915', 'SET:100.0.9.95:1.0.9.25:80:2012916', 'SET:100.0.9.95:1.0.9.25:80:2012917', 'SET:100.0.9.95:1.0.9.25:80:multi']

compensated_variance: 31178
online_variance: 156444
online_variance_n: 156362
web_std_dev (0.01108764481808884, 0.48254481793081905)
nslookup details: Server:               208.67.222.222
Address:        208.67.222.222#53
Finished

** server can't find 25.9.0.1.in-addr.arpa.: NXDOMAIN



[18, 21, 21, 837, 21, 22, 21, 836, 21, 21, 21, 837, 21, 21, 21, 837, 21, 21, 21, 837, 21, 21, 21, 837, 21, 21, 21, 837, 21, 21, 21,
837, 21, 21, 21, 837, 21, 21, 21, 837, 21, 21, 21, 837, 21, 21, 22, 836, 21, 21, 21, 837, 21, 21, 21, 837, 21, 21, 21, 837, 21, 21,
21, 837, 21, 21, 21, 837, 21, 21, 21, 837, 21, 21, 21, 837, 21, 21, 21, 837, 21, 21, 21, 837, 21, 21, 21, 837, 21, 21, 21, 837, 21,
21, 21, 837, 21, 21, 21, 837, 21, 21, 21, 837, 21, 21, 21, 837, 21, 21, 21, 837, 21, 21, 21, 837, 20, 21, 21, 838, 21, 21, 21, 837,
21, 21, 21, 837, 20, 22, 21, 837, 21, 21, 21, 837, 21, 21, 21, 837, 21, 21, 21, 837, 20, 21, 21, 838, 21, 21, 21, 837, 20, 22, 21, 836, 22, 21, 21,
836, 21, 21, 22, 836, 22, 21, 21, 836, 22, 21, 21, 836, 21, 21, 21, 837, 22, 21, 21, 836, 21, 21, 21, 837, 21, 22, 21, 836, 21, 22, 21, 836, 21, 22,
21, 836, 21, 22, 21, 836, 21, 24, 18, 837, 21, 22, 21, 837, 20, 21, 22, 836, 21, 21, 21, 837, 22, 21, 21, 836, 21, 21, 21, 837, 21, 22, 20, 837, 21,
21, 21, 837, 21, 21, 22, 836, 21, 21, 21, 837, 21, 21, 21, 837, 21, 22, 21, 836, 21, 22, 21, 836, 21, 22, 21, 836, 21, 21, 21, 837, 21, 21, 21, 837,
21, 21, 21, 837, 21, 21, 22, 836, 21, 22, 21, 836, 21, 21, 21, 837, 21, 21, 22, 836, 21, 22, 21, 836, 21, 21, 21, 837, 21, 21, 21, 837, 21, 21, 21,
837, 21, 21, 21, 837, 21, 21, 21, 837, 21, 22, 21, 836, 21, 21, 21, 837, 21, 21, 21, 837, 21, 21, 21, 837, 21, 21, 21, 837, 21, 22,
21, 836, 21, 21, 22, 836, 21, 21, 22, 836, 21, 21, 21, 837, 21, 21, 21, 837, 21, 21, 21, 837, 21, 21, 21, 837, 21, 21, 22, 836, 21, 21, 21, 837, 21,
21, 22, 836, 21, 21, 21, 837, 21, 21, 21, 837, 21, 21, 21, 837, 21, 21, 21, 837, 21, 21, 24, 834, 21, 21, 21, 837, 21, 21, 21, 837, 21, 21, 21, 837,
21, 21, 21, 837, 21, 21, 21, 837, 21, 21, 21, 837, 21, 21, 21, 837, 21, 21, 21, 21, 21, 21, 837, 21, 21, 21, 837, 21, 21, 21, 837, 21, 21, 21, 838,
21, 21, 21, 837, 21, 21, 21, 837, 21, 21, 21, 837, 21, 22, 21, 836, 21, 21, 21, 837, 21, 21, 21, 838, 21, 21, 21, 837, 21, 21, 21, 837, 21, 21, 21,
```

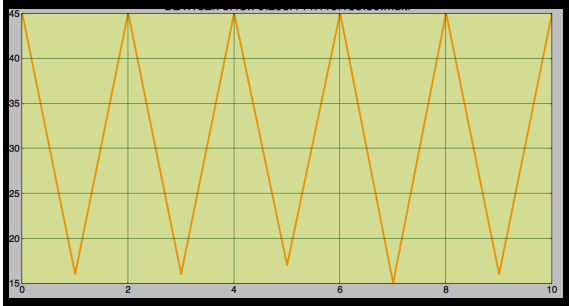
Plot Text OUTPUT example

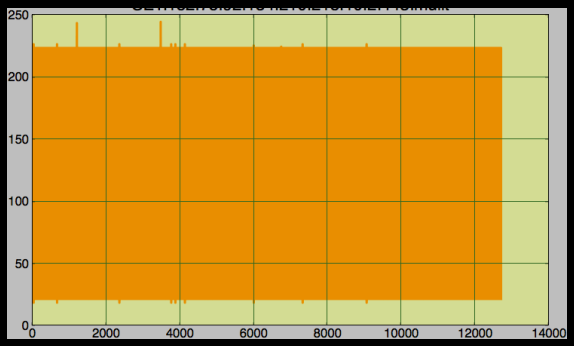
Graphing 1

Findings
Dialing the tolerances to each network is important
If you open the tolerance to include traffic just outside the statistical significant will leads to interesting results

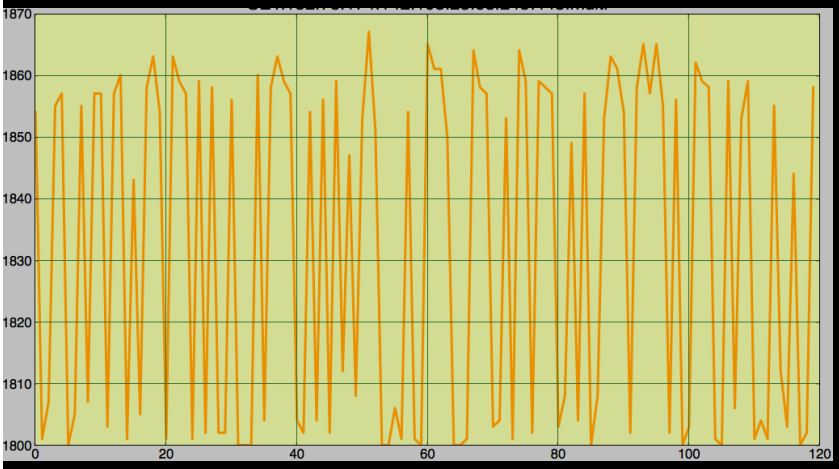


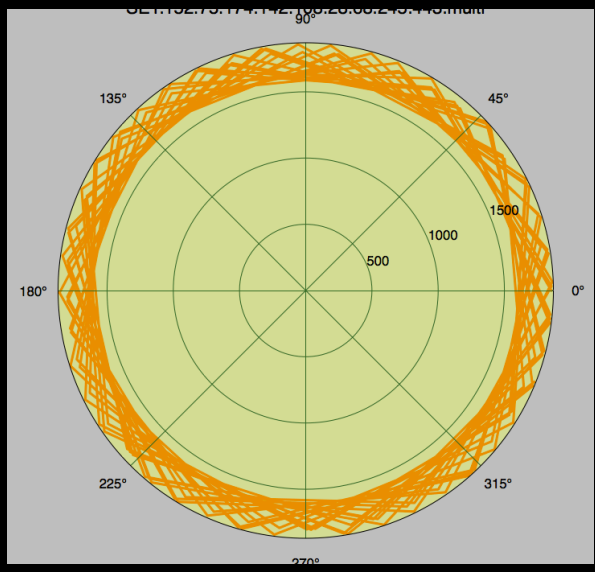






timing of a sample beacon





Considerations

- 1. Tune variables to a specific network
  - a. Host count
  - b. vistors
- 2. Outlier reject may exclude useful results
- 3. Results should include domain results
- 4. Excluding trusted sources saves time
  - a. Trusted list requires management
- 5. Continuous collection and periodic analysis needs more testing
  - a. Expiration of data (production)
  - b. Scheduled analysis
    - i. Output top list
      - 1. include graphical output
  - c. Require periodic flush of the database

Conclusion

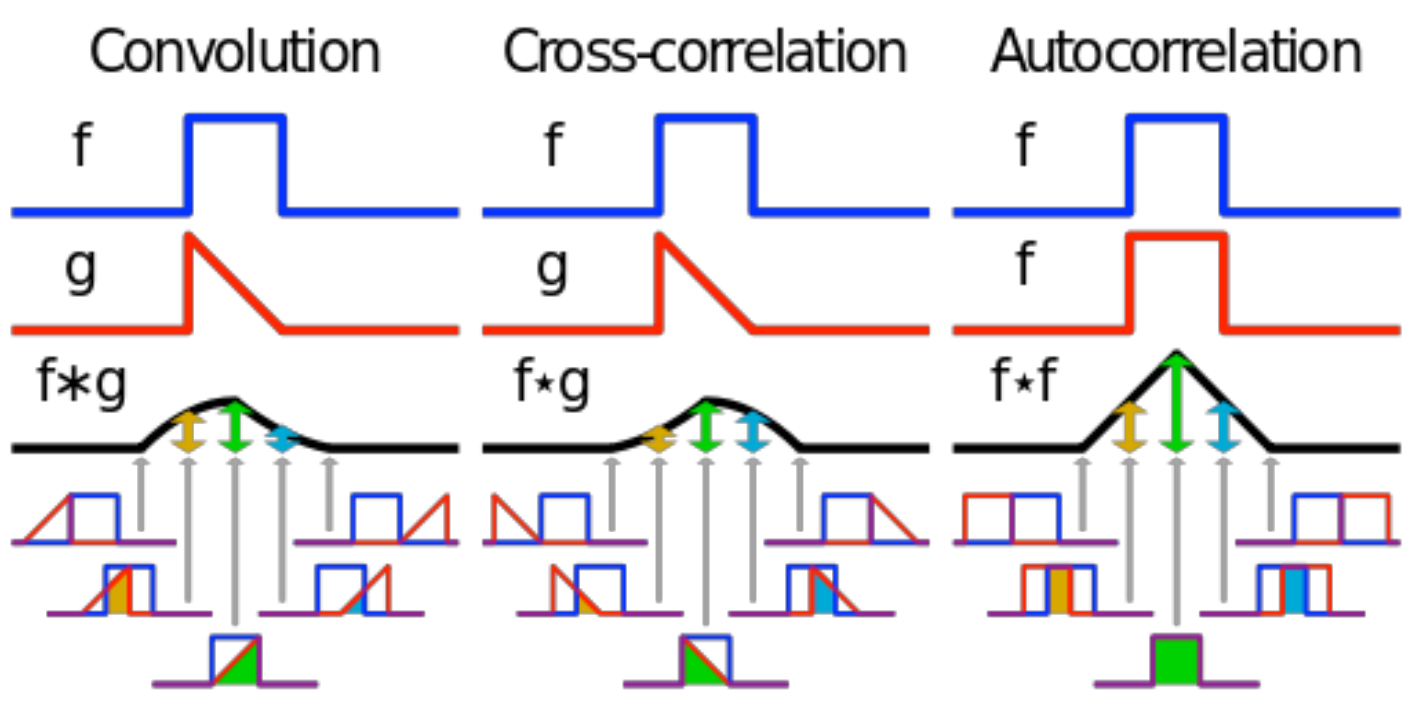
1. Timing is a signature
2. Expanding beacon detection to include payload analysis seems useful
3. Full packet capture can assist in validating threats
4. Host inspection is the best way to validate threats
5. Expand tracking to include DNS
6. Variable timing is difficult but not impossible to include in the analysis
7. Easy to include nslookup and whois results in our dataset

Tools

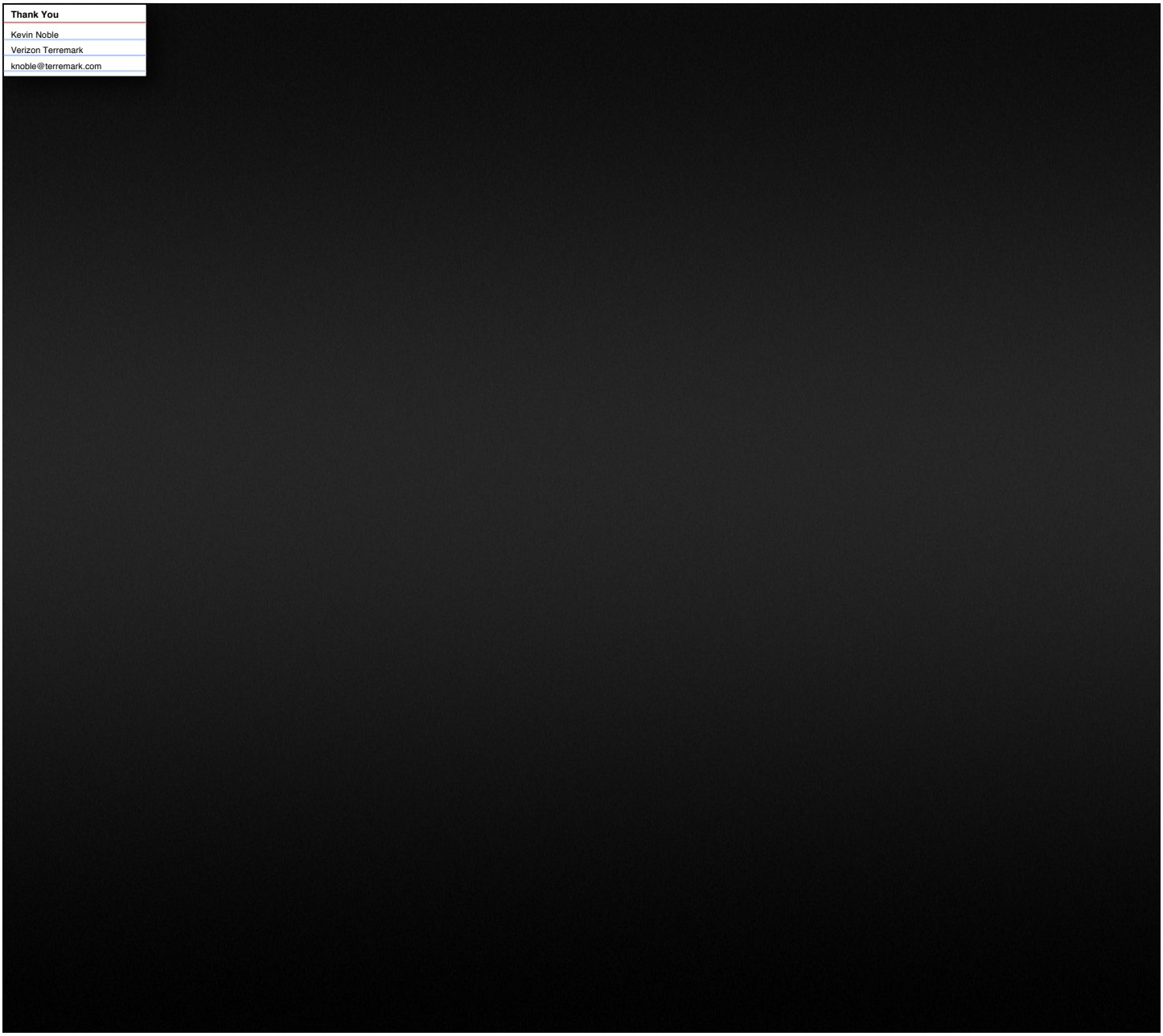
1. Flow collection
   - a. Code http://code.google.com/p/beaconbits
   - b. ARGUS
     - i. http://www.qosient.com/argus/
2. Dev Code
   - a. Python 2.7.1
     - i. Library for Redis
       - 1. https://github.com/andymccurdy/redis-py
     - ii. Library for Stats
       - 1. http://www.jstor.org/stable/1266577
       - 2. NUMPY
       - 3. MATPLOTLIB
   - b. IDE editor
     - i. Komodo IDE V2
3. Database
   - a. Redis 2.5.11 (00000000/0) 64 bit
     Running in stand alone mode
     http://redis.io
4. Presentation
   - a. CURIO
     - i. http://www.zengobi.com/products/curio

Future considerations

1. Release a production capable version (with enough public interest)
2. Release a stand alone version (no redis required, just reads flows and outputs)
3. Include the use of exclusion list (trust / clean list)
4. Time series analysis with autocorrelation



Untitled


Thank You
Kevin Noble
Verizon Terremark
knoble@terremark.com