Baseline the Network with Zeek
Who am I?

• Network defender, analyst and integrator

• Working with Bro/Zeek for about 10 years

• Experience deploying, operating and leveraging Zeek in many environments, large and small

• Always looking for novel ways to use Zeek to solve network monitoring problems
Agenda

• The Problem
• What are baselines?
• Discuss a new Zeek module for creating them
• Instrument traffic analysis techniques
• Using baselines
• Other considerations
The A Problem

• Network Defenders Dilemma – you must understand normal in order to identify abnormal

• This is profoundly difficult, especially for new analysts

• Is what is happening now normal compared what happened 10 minutes ago? yesterday? last week?

• Given all the data available, where do you start?
More about the data

• Most protocol metadata is qualitative
  • IP address, User Agent, URL, Domain, Port Numbers

• Byte and Packet counters are quantitative

• Other quantitative measures:
  • Duration, interval, rate

• Without additional context still difficult to use to gain an understanding of normal
What are baselines?
What are baselines? really...

A minimum or starting point used for comparisons
How can we create and use one?

• Make quantitative observations that describe host behavior

• Record those observations in a standard, easily consumed format

• Analyze the data, look for patterns and deviations
NetBase
(Network Baselineer)
Netbase at a high level

• For each *monitored* IP address record observations that describe attributes and behaviors - *observables*

• Accrue these observations for a set period of time – 5 *mins*

• At the end of the interval, log a summary of the observations
Netbase Structure

Zeek Logger

Zeek Proxy

Zeek Worker

Zeek Proxy

Zeek Worker

Zeek Worker

observations

observables
the observation record

type observation: record {
    address: addr &log &optional;
    starttime: time &log &optional;
    endtime: time &log &optional;
} &redef;
the observations table

```json
{
    [192.168.10.9] = [address=192.168.10.9,
    starttime=1570474522.835633,
    endtime=<uninitialized>,
    observables...],
    [192.168.10.15] = [address=192.168.10.15,
    starttime=1570474493.419398,
    endtime=<uninitialized>,
    observables...]}
```
type observable: record {
    name: string;
    val: string &optional;
};

Name corresponds to new fields added to observations record
the *SEND* function

```cpp
function SEND(ip: addr, obs: set[observable])
{
    Cluster::publish_hrw(Cluster::proxy_pool,
        ip,
        add_observables,
        ip,
        obs);
    event Netbase::add_observables(ip, obs);
}
```
the netbase log stream

{
    "address": "192.168.10.3",
    "starttime": "2019-10-07T19:10:06.652734Z",
    "endtime": "2019-10-07T19:15:09.413167Z",
    "ext_client_cnt": 0,
    "ext_host_cnt": 1,
    "ext_port_cnt": 1,
    "ftp_auth_failures": 0,
    "ftp_failed_auth_attempts": 0,
    ...
}
the netbase stats log stream

```json
{
  "ts": "2019-10-07T18:59:18.476335Z",
  "node_id": "proxy-2",
  "addr_cnt": 6,
  "table_size": 54624
}
```
protocol-specific modules

Lets talk observables
Observable types

- Currently using two types:
  - Counters of occurrences
  - Distinct value counts

- Every time an IP’s comms meet a condition, increment a counter

- Distinct counts record the number of instances of some thing

- Plan to add others like: mean, max and min
## Conn observables

<table>
<thead>
<tr>
<th>int_port_cnt</th>
<th>out_orig_conns</th>
<th>int_orig_conns</th>
</tr>
</thead>
<tbody>
<tr>
<td>int_host_cnt</td>
<td>out_succ_conns</td>
<td>int_succ_conns</td>
</tr>
<tr>
<td>ext_port_cnt</td>
<td>out_rej_conns</td>
<td>int_rej_conns</td>
</tr>
<tr>
<td>ext_host_cnt</td>
<td>out_to_port#</td>
<td>int_resp_conns</td>
</tr>
</tbody>
</table>
DNS observables

dns_as_server    dns_auth_answers    dns_ext_rr_cnt

dns_as_client    dns_recur_answers    dns_int_rr_cnt

dns_nxdomain_sent    dns_nxdomain_rcvd
HTTP observables

http_as_server  http_as_client  http_post_sent

http_post_recvd  http_get_sent  http_get_recvd

http_400_recvd  http_400_sent
Back to the Baselines

• By creating a running record of these observations, per IP, you are in effect creating a baseline

• Point in time observations that can be compared manually, visually or statistically

• Compare observations for a given IP to previous observations

• Compare observations for multiple IP’s at once

• Compare across other dimensions using asset information
Let's see it!
Y Axis = HTTP GET Received, X Axis = HTTP POST Received
All observations are for a single web server
FTP Brute force

Y Axis = FTP Client Failed to Authenticate, X Axis = FTP Server Responded with Auth Failure
Port Scanning

Y Axis = Internal Host Count, X Axis = Internal Port Count, Bubble Size = Rejected Conn count
Gotchas and limitations

- Transient hosts, devices that aren’t always connected
- DHCP - IP addresses may move around
- Large networks, lots of IP addresses in use
- May not be suitable for every host in the environment
Does it scale?
IP’s Tracked by Proxy
Observations: Table Size by Proxy
Future Work

• Get it cleaned up and released as a Zeek 3.0 package
• Add new observable types: mean, max, min
• Add more protocol-specific observables
• Analytics
Conclusion

- Network baselines are a real thing with practical application in cyber network defense
- Many ways to categorize host network behavior
- Zeek is a great tool for turning behavioral observations into quantitative data
Thank You!

adam@nimbuscyber.io