Bro SumStats Framework Exercises

The SumStats framework is a continuation and rename of the old Metrics framework from Bro 2.0 and 2.1. The API and capabilities have been greatly expanded and improved. A major goal for SumStats has been to provide advanced statistical and summarization capabilities in a way that is both relatively easy to use and can perform on clusters.

Part 1: Analysis and Thresholding

For all of these exercises we'll be using the exercise-traffic.pcap file.

Exercise

First we need to observe something so we need to consider the very base thing we want to measure. DNS query names by the requester perhaps? Copy the following code into a file named sumstats-1.bro.

```
bro

event dns_request(c: connection, msg: dns_msg, query: string,
    qtype: count, qclass: count)
    {
    if ( c$id$resp_p == 53/udp && query != "" )
        SumStats::observe("dns.lookup", [$host=c$id$orig_h], [
            $str=query]);
    }
```

Running this script as it is won't provide any output, we haven't instructed the SumStats framework to do anything with those observations. In the same script, below the previous chunk of code, paste this code:

```
bro

event bro_init()
    {
    local r1 = SumStats::Reducer($stream="dns.lookup", $apply=
        set(SumStats::UNIQUE));
    SumStats::create([$name="dns.requests.unique",
                      $epoch=6hr,
                      $reducers=set(r1),
                      $epoch_result(ts: time, key: SumStats::
                          Key, result: SumStats::Result) =
                        {
                        local r = result["dns.lookup"];
```


```
                        print fmt("%s did %d total and %d
                            unique DNS requests in the last 6
                            hours.",
                                    key$host, r$num, r$unique);
                        },
                    $epoch_finished(ts: time) =
                        {
                        print "-----------";
                        }]);
    }
```

This script creates a reducer which collects unique values of the third argument of the "observe" call above (DNS queries). The reducer is then added to the SumStat we're creating which has a measurement epoch of 6 hours. At the end of the 6 hour period, the "epoch_result" callback will be called once for each result value being tracked. There will be a result value for each "Key" which is the second argument in the observe call above. Once all of the keys have been processed through the epoch_result callback, the epoch_finished callback will be called.

We should now be able to take a look at our result from this script.

```
bro -r exercise-traffic.pcap sumstats-1.bro
```

Exercise

Now we can go a slightly different direction and set a threshold on the number of unique DNS requests. Use the same bit of code from above for doing the observation but replace the bro_init event handler from the previous exercise with the following code into a new file named sumstats-2.bro.

```
bro

event bro_init()
    {
    local r1 = SumStats::Reducer($stream="dns.lookup", $apply=
        set(SumStats::UNIQUE));
    SumStats::create([$name="dns.thresholding",
                      $epoch=6hrs,
                      $reducers=set(r1),
                      $threshold_val(key: SumStats::Key, result
                          : SumStats::Result) =
                        {
                        return result["dns.lookup"]$unique+0.0;
                        },
                      $threshold=150.0,
                      $threshold_crossed(key: SumStats::Key,
                          result: SumStats::Result) =
                        {
                        print fmt("%s did more than 150 unique
                            requests!", key$host);
                        }]);
    }
```

You can see that this script is slightly different in that it doesn't have the epoch_result or epoch_finished callbacks. Instead it has several threshold related fields filled out.

"threshold_val" is a callback that must be provided when doing SumStat thresholding and it needs to return a double value of the current value you want your threshold applied to. "threshold" is just a double that you want the threshold to be. "threshold_crossed" is a callback that is called when a threshold has been crossed.

Now run

```
bro -r exercise-traffic.pcap sumstats-2.bro
```

Exercise

One final small change we can make is to do a threshold on a ratio of unique to total DNS requests. Using the same "observe" call we've been using this whole time, now use the following SumStat to set a threshold on the number of distinct DNS lookups being performed by a host by comparing the number of unique requests to the total number of requests. Name this file sumstat-3.bro.

```
bro

event bro_init()
    {
    local r1 = SumStats::Reducer($stream="dns.lookup", $apply=
        set(SumStats::UNIQUE));
    SumStats::create([$name="dns.distinct.thresholding",
                      $epoch=6hrs,
                      $reducers=set(r1),
                      $threshold_val(key: SumStats::Key, result
                          : SumStats::Result) =
                        {
                        local r = result["dns.lookup"];
                        # We want at least 50 DNS requests
                            before even applying this
                        # ratio based threshold.
                        if ( r$num < 50 )
                            return 0.0;

                        return (r$unique+0.0)/(r$num+0.0);
                        },
                      $threshold=0.95,
                      $threshold_crossed(key: SumStats::Key,
                          result: SumStats::Result) =
                        {
                        local r = result["dns.lookup"];
                        print fmt("%.0f%% or more of the %d DNS
                             requests made by %s are distinct.",
                                  ((r$unique+0.0)/(r$num+0.0)
                                      *100), r$num, key$host);
                        }]);
    }
```

Now run this script:

```
bro -r exercise-traffic.pcap sumstats-3.bro
```

Part 2: Probabilistic Top-K

There are times where the top most frequently seen things are something that is interesting from a performance monitoring, network tuning, or even security perspective. Bro 2.2's new probabilistic Top-K support can provide that data in an easy to consume way.

Exercise

If you want to know the top 10 names being requested over DNS on a network that would normally be quite difficult, but with the SumStats framework it's actually a relatively small bit of code to get that data on a single Bro process or to get the same result on a large cluster.

Paste the following code sample into a file named sumstats-4.bro.

```
event bro_init()
    {
    local r1 = SumStats::Reducer($stream="dns.lookups", $apply=
        set(SumStats::TOPK), $topk_size=50);
    SumStats::create([$name="top_dns_lookups",
                      $epoch=12hrs,
                      $reducers=set(r1),
                      $epoch_result(ts: time, key: SumStats::
                          Key, result: SumStats::Result) =
                        {
                        local r = result["dns.lookups"];
                        local s: vector of SumStats::
                            Observation;
                        s = topk_get_top(r$topk, 10);
                        print fmt("Top 10 DNS requests by %s
                            for %D through %D", key$host,
                            r$begin, r$end);
                        for ( i in s )
                            {
                            if ( i == 10 )
                                break;

                            print fmt("   Name: %s (estimated
                                count: %d)",
                                          s[i]$str, topk_count(
                                              r$topk, s[i]));
                            }
                            # Add an extra line for nice
                                formatting.
                            print "";
                        }
                      ]);
    }

event dns_request(c: connection, msg: dns_msg, query: string,
    qtype: count, qclass: count)
    {
    if ( c$id$resp_p == 53/udp && query != "" )
```


```
        SumStats::observe("dns.lookups", [$host=c$id$orig_h], [
            $str=query]);
    }
```

This is very similar to the code samples from before and the main difference to pay attention to is the reducer. You can see the TOPK algorithm is being applied to observations being fed into the reducer.

Now run

```
bro -r exercise-traffic.pcap sumstats-4.bro
```

Exercise

The previous example was nice because it showed calculating lots of separate Top-10 results. Something a bit more useful in live network traffic might be to calculate the Top-10 DNS requests for everything in the entire network.

Paste the folowing code into a file named sumstats-5.bro.

```
event bro_init()
    {
    local r1 = SumStats::Reducer($stream="dns.lookups", $apply=
        set(SumStats::TOPK), $topk_size=50);
    SumStats::create([$name="top_dns_lookups",
                      $epoch=12hrs,
                      $reducers=set(r1),
                      $epoch_result(ts: time, key: SumStats::
                          Key, result: SumStats::Result) =
                        {
                        local r = result["dns.lookups"];
                        local s: vector of SumStats::
                            Observation;
                        s = topk_get_top(r$topk, 10);
                        print fmt("Top 10 DNS requests for %D
                            through %D", r$begin, r$end);
                        for ( i in s )
                            {
                            if ( i == 10 )
                                break;

                            print fmt("   Name: %s (estimated
                                count: %d)", s[i]$str,
                                topk_count(r$topk, s[i]));
                            }
                            # Add an extra line for nice
                                formatting.
                            print "";
                        }]);
    }

event dns_request(c: connection, msg: dns_msg, query: string,
    qtype: count, qclass: count)
    {
    if ( c$id$resp_p == 53/udp && query != "" )
        SumStats::observe("dns.lookups", [], [$str=query]);
    }
```

Now run

```
bro -r exercise-traffic.pcap sumstats-5.bro
```