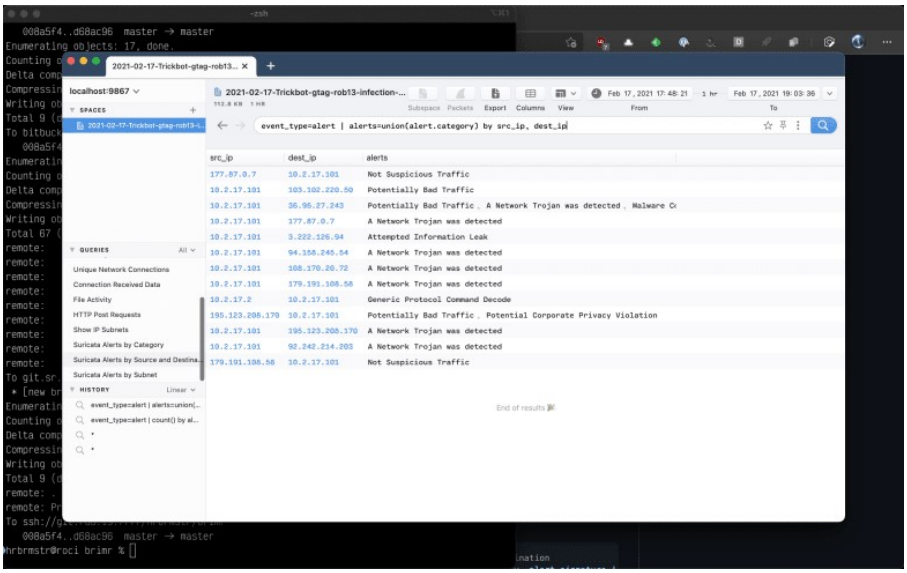
Brim Security maintains a free, Electron-based desktop GUI for exploration of PCAPs and select cybersecurity logs:



along with a broad ecosystem of tools which can be used independently of the GUI.

The standalone or embedded `zqd` server, as well as the `zq` command line utility let analysts run ZQL (a domain-specific query language) queries on cybersecurity data sources.

The Brim team maintains a Python module that is capable of working with the `zqd` HTTP API and my nascent {brimr}[gitea|gh|gl|bb] R package provides a similar API structure to perform similar operations in R, along with a wrapper for the `zq` commmand line tool.

PCAPs! In! Spaaaaacce[s]!

Brim Desktop organizes input sources into something called “spaces”. We can check for available spaces with `brim_spaces()`:

```
library(brimr)
library(tibble)

brim_spaces()
##                                   id
name
## 1 sp_1p6pwLgtsESYBTHU9PL9fcl2iBn 2021-02-17-Trickbot-gtag-rob13-
infection-in-AD-environment.pcap
##
data_path storage_kind
## 1 file:///Users/demo/Library/Application%20Support/Brim/
data/spaces/sp_1p6pwLgtsESYBTHU9PL9fcl2iBn    filestore
```

This single space availble is a sample capture of Trickbot

Let’s profile the network connections in this capture:

```
# ZQL query to fetch Zeek connection data
zql1 <- '_path=conn | count() by id.orig_h, id.resp_h, id.resp_p | sort
```

```
id.orig_h, id.resp_h, id.resp_p'

space <- "2021-02-17-Trickbot-gtag-rob13-infection-in-AD-
environment.pcap"

r1 <- brim_search(space, zql1)

r1
## ZQL query took 0.0000 seconds; 384 records matched; 1,082 records
read; 238,052 bytes read

(r1 <- as_tibble(tidy_brim(r1)))
## # A tibble: 74 x 4
##    orig_h      resp_h       resp_p count
##
##  1 10.2.17.2   10.2.17.101  49787      1
##  2 10.2.17.101 3.222.126.94 80         1
##  3 10.2.17.101 10.2.17.1    445        1
##  4 10.2.17.101 10.2.17.2    53        97
##  5 10.2.17.101 10.2.17.2    88        27
##  6 10.2.17.101 10.2.17.2    123        5
##  7 10.2.17.101 10.2.17.2    135        8
##  8 10.2.17.101 10.2.17.2    137        2
##  9 10.2.17.101 10.2.17.2    138        2
## 10 10.2.17.101 10.2.17.2    389       37
## # … with 64 more rows
```

Brim auto-processed the PCAP into Zeek log format and `_path=conn` in query string indicates that's where we're going to perform further data operations (the queries are structured a bit like `jq` filters). We then ask Brim/`zqd` to summarize and sort source IP, destination IP, and port counts. {brimr} sends this query over to the server. The raw response is a custom data structure that we can turn into a tidy data frame via `tidy_brim()`.

We can do something similar with the Suricata data that Brim also auto-processes for us:

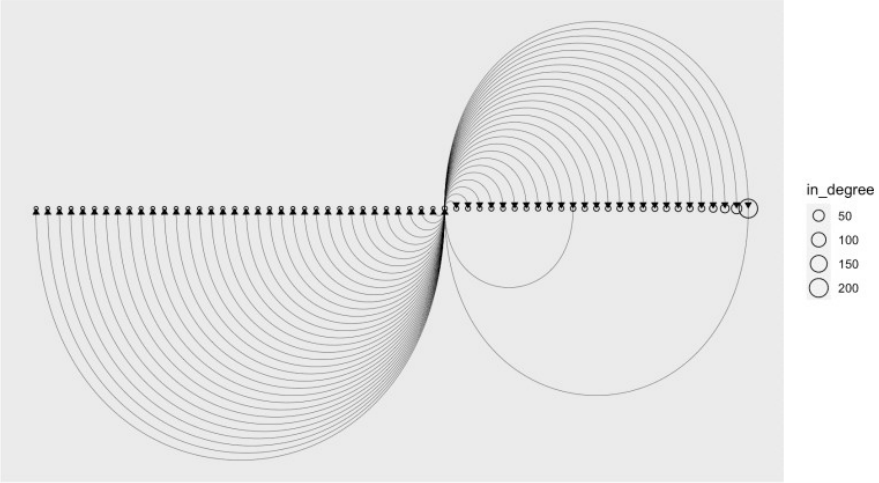
```
# Z query to fetch Suricata alerts including the count of alerts per
source:destination
zql2 <- "event_type=alert | count() by src_ip, dest_ip, dest_port,
alert.severity, alert.signature | sort src_ip, dest_ip, dest_port,
alert.severity, alert.signature"

r2 <- brim_search(space, zql2)

r2
## ZQL query took 0.0000 seconds; 47 records matched; 870 records read;
238,660 bytes read

(r2 <- (as_tibble(tidy_brim(r2))))
## # A tibble: 35 x 6
##    src_ip     dest_ip    dest_port severity signature
count
##
```

```
##  1 10.2.17.2  10.2.17.1…     49674        3 SURICATA Applayer Detect
protocol only one direction                      1
##  2 10.2.17.2  10.2.17.1…     49680        3 SURICATA Applayer Detect
protocol only one direction                      1
##  3 10.2.17.2  10.2.17.1…     49687        3 SURICATA Applayer Detect
protocol only one direction                      1
##  4 10.2.17.2  10.2.17.1…     49704        3 SURICATA Applayer Detect
protocol only one direction                      1
##  5 10.2.17.2  10.2.17.1…     49709        3 SURICATA Applayer Detect
protocol only one direction                      1
##  6 10.2.17.2  10.2.17.1…     49721        3 SURICATA Applayer Detect
protocol only one direction                      1
##  7 10.2.17.2  10.2.17.1…     50126        3 SURICATA Applayer Detect
protocol only one direction                      1
##  8 10.2.17.1… 3.222.126…        80        2 ET POLICY curl User-
Agent Outbound                                        1
##  9 10.2.17.1… 36.95.27.…       443        1 ET HUNTING Suspicious
POST with Common Windows Process Names - Possib…     1
## 10 10.2.17.1… 36.95.27.…       443        1 ET MALWARE
Win32/Trickbot Data Exfiltration                              1
## # … with 25 more rows
```

Finally, for this toy example, we'll also generate a visual overview of these connections:

```
library(igraph)
library(ggraph)
library(tidyverse)

gdf <- count(r1, orig_h, resp_h, wt=count)

count(gdf, node = resp_h, wt=n, name = "in_degree") %>%
  full_join(
    count(gdf, node = orig_h, name = "out_degree")
  ) %>%
  mutate_at(
    vars(in_degree, out_degree),
    replace_na, 1
  ) %>%
  arrange(in_degree) -> vdf

g <- graph_from_data_frame(gdf, vertices = vdf)

ggraph(g, layout = "linear") +
  geom_node_point(
    aes(size = in_degree), shape = 21
  ) +
  geom_edge_arc(
    width = 0.125,
    arrow = arrow(
      length = unit(5, "pt"),
      type = "closed"
    )
```

```
)
```



We can also process log files directly (i.e. without any server) with `zq_cmd()`:

```
zq_cmd(
  c(
    '"* | cut ts,id.orig_h,id.orig_p"', # note the quotes
    system.file("logs", "conn.log.gz", package = "brimr")
   )
 )
##           id.orig_h id.orig_p                          ts
##   1:  10.164.94.120     39681 2018-03-24T17:15:21.255387Z
##   2:    10.47.25.80     50817 2018-03-24T17:15:21.411148Z
##   3:    10.47.25.80     50817 2018-03-24T17:15:21.926018Z
##   4:    10.47.25.80     50813 2018-03-24T17:15:22.690601Z
##   5:    10.47.25.80     50813 2018-03-24T17:15:23.205187Z
##  ---
## 988: 10.174.251.215     33003 2018-03-24T17:15:21.429238Z
## 989: 10.174.251.215     33003 2018-03-24T17:15:21.429315Z
## 990: 10.174.251.215     33003 2018-03-24T17:15:21.429479Z
## 991:  10.164.94.120     38265 2018-03-24T17:15:21.427375Z
## 992: 10.174.251.215     33003 2018-03-24T17:15:21.433306Z
```

FIN

This package is less than 24 hrs old (as of the original blog post date) and there are still a few bits missing, which means y'all have the ability to guide the direction it heads in. So kick the tyres and interact where you're most comfortable.