STA141B—Assignment 2

Filip Wilhelm Sjostrand

2023-05-01

Create a data.frame

First of all we have to separate each of the five log files. We should try to find the pattern suggested in the prompt *# log-file-name*. By using the `grepl()` + `table()` we iterated over different possible regex until we found a match of 5 **TRUE**. Eventually we assumed that only log file names stated with '#'. Indeed the file names appear to be extracted below. We are thrown an error saying "incomplete final line..." which can be ignored in our case (Pon, 2011).

```
# Packages ----------
library(readr)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
source("/Users/filipsjostrand/Documents/UC Davis/Courses/STA 141B/assignments/2/functions.R")

# Get Data ----------
dir <- "/Users/filipsjostrand/Documents/UC Davis/Courses/STA 141B/data/asssign2"
log_file <- file.path(dir, "MergedAuth.log")
file.exists(log_file)
```

```
## [1] TRUE
```

```
ll <- suppressWarnings(readLines(log_file))
head(ll)
```

```
## [1] ""
## [2] "# auth.log"
## [3] "Nov 30 06:39:00 ip-172-31-27-153 CRON[21882]: pam_unix(cron:session): session closed for user r
## [4] "Nov 30 06:47:01 ip-172-31-27-153 CRON[22087]: pam_unix(cron:session): session opened for user r
## [5] "Nov 30 06:47:03 ip-172-31-27-153 CRON[22087]: pam_unix(cron:session): session closed for user r
## [6] "Nov 30 07:07:14 ip-172-31-27-153 sshd[22116]: Connection closed by 122.225.103.87 [preauth]"
```

```r
# Identify Pattern ----------
rx <- "^#"
names <- grepl(rx, ll)
table(names)
```

```
## names
## FALSE  TRUE
## 99963     5
```

```r
ll[names]
```

```
## [1] "# auth.log"                   "# auth2.log"
## [3] "# loghub/Linux/Linux_2k.log" "# loghub/Mac/Mac_2k.log"
## [5] "# loghub/OpenSSH/SSH_2k.log"
```

After some trial and error we decide to split the tables by its file names, work on them separately and merge the final result. From the split_tables function we find that it indeed split into 5.

`length(ll)` gives 99968 and removing 5 file names and the empty row at the top imply that the total length of each table should equal 99962, which was true. However, that also included the empty row that appears before each file name. Thus, removing those should equal to 99958, which it does.

```r
# Remove file names & create data.frames
start_index <- grep(rx, ll)
tt <- split_tables(ll, start_index)
sum(lengths(tt))
```

```
## [1] 99958
```

```r
tail(tt[[1]])
```

```
## [1] "Dec 31 22:25:45 ip-172-31-27-153 sshd[7996]: Connection closed by 218.2.0.133 [preauth]"
## [2] "Dec 31 22:26:20 ip-172-31-27-153 sshd[7998]: Connection closed by 218.2.0.133 [preauth]"
## [3] "Dec 31 22:27:07 ip-172-31-27-153 sshd[8001]: Connection closed by 218.2.0.133 [preauth]"
## [4] "Dec 31 22:27:48 ip-172-31-27-153 sshd[8003]: Invalid user admin from 218.2.0.133"
## [5] "Dec 31 22:27:48 ip-172-31-27-153 sshd[8003]: input_userauth_request: invalid user admin [preauth]"
## [6] "Dec 31 22:27:48 ip-172-31-27-153 sshd[8003]: Connection closed by 218.2.0.133 [preauth]"
```

lets start parsing, beginning with the date-time. We assume that the date is formatted equally across. Iterating over different regular expressions we found a table with no **FALSE**. `rx_tester()` returns nothing if only TRUE matches are found. Indeed the all the dates match the following pattern. We now want to transform into POSIXct. In order to do so we have to add a dummy year. Given no error thrown, we can confirm it is a sucessfull format.

```r
# Parsing date-time ----------
rx <- "[A-Z][a-z]{2} +[0-9]{1,2} [0-9]{2}:[0-9]{2}:[0-9]{2}"
res <- parser(tt, rx)
dummy <- lapply(res[[1]], paste, "2023")
dt <- lapply(dummy, as.POSIXct, format = "%b %d %H:%M:%S %Y")
```

Further, we want to parse the logging host. We find it simpler to remove the previously successfully parsed parts. Quick visual check confirms successful removal of the date time from the raw file. We are assuming that we have a space between date-time, logging host, and app, with no spaces within logging host. using `rx2` we find that `rx_tester()` return nothing thus indicating sucessfull parsing. The sample does not suggest otherwise.

```
# Parsing logging host ----------
rx2 <- "^ ([a-zA-Z0-9]|-)+ "
res2 <- parser(res[[2]], rx2)
log_host <- lapply(res2[[1]], trimws)
lapply(log_host, sample, 10)
```

```
## [[1]]
##  [1] "ip-172-31-27-153" "ip-172-31-27-153" "ip-172-31-27-153" "ip-172-31-27-153"
##  [5] "ip-172-31-27-153" "ip-172-31-27-153" "ip-172-31-27-153" "ip-172-31-27-153"
##  [9] "ip-172-31-27-153" "ip-172-31-27-153"
##
## [[2]]
##  [1] "ip-10-77-20-248" "ip-10-77-20-248" "ip-10-77-20-248" "ip-10-77-20-248"
##  [5] "ip-10-77-20-248" "ip-10-77-20-248" "ip-10-77-20-248" "ip-10-77-20-248"
##  [9] "ip-10-77-20-248" "ip-10-77-20-248"
##
## [[3]]
##  [1] "combo" "combo" "combo" "combo" "combo" "combo" "combo" "combo" "combo"
## [10] "combo"
##
## [[4]]
##  [1] "authorMacBook-Pro"         "calvisitor-10-105-162-178"
##  [3] "authorMacBook-Pro"         "authorMacBook-Pro"
##  [5] "authorMacBook-Pro"         "authorMacBook-Pro"
##  [7] "calvisitor-10-105-162-178" "calvisitor-10-105-163-202"
##  [9] "calvisitor-10-105-162-105" "calvisitor-10-105-162-105"
##
## [[5]]
##  [1] "LabSZ" "LabSZ" "LabSZ" "LabSZ" "LabSZ" "LabSZ" "LabSZ" "LabSZ" "LabSZ"
## [10] "LabSZ"
```

For the app and PID we shall approach it in a two step process. First, extracting it as one chunk, then split within, also indicating missing PIDs. We begin by removing the previously successfully parsed section. `head(tt3[[i]])` suggest successful removal. Visually it appears as if the app + PID chunk always ends at ":". It appears as a good start, however, we have two cases to handle: some PID are missing, and some PID are followed by a chunk that belongs to the message part. By simply adding that we should read until the first ocurance of either ":" or "]" we solved it.

```
# Extracting PID and app ----------
rx3 <- "^.*?(\\]|:)"
res3 <- parser(res2[[2]], rx3)
messages <- lapply(res3[[2]], trimws)
lapply(res3[[1]], sample, 5)
```

```
## [[1]]
## [1] "sshd[10096]" "sshd[28736]" "sshd[11398]" "sshd[19220]" "sshd[24126]"
##
```

```
## [[2]]
## [1] "sudo:"                "chpasswd[32282]"      "CRON[17837]"
## [4] "chpasswd[682]"        "systemd-logind[1118]"
##
## [[3]]
## [1] "sshd(pam_unix)[20969]" "ftpd[32330]"           "ftpd[24486]"
## [4] "sshd(pam_unix)[15696]" "sshd(pam_unix)[30739]"
##
## [[4]]
## [1] "kernel[0]"          "kernel[0]"          "CalendarAgent[279]"
## [4] "kernel[0]"          "kernel[0]"
##
## [[5]]
## [1] "sshd[24421]" "sshd[25098]" "sshd[24509]" "sshd[24363]" "sshd[25326]"
```

Now, any entry that does not end with "]" need to be pasted with "[NA]". A first approach is to simply sub ":" with "[NA]"—which turned out good. Then we split on "[...]"

```
# Adding NAs
pid_app <- lapply(res3[[1]], gsub, pat = ":", repl = "[NA]")
pid_app <- lapply(pid_app, strsplit, split = "\\[|\\]", perl = T)
lapply(pid_app, sample, 2)
```

```
## [[1]]
## [[1]][[1]]
## [1] "sshd" "7432"
##
## [[1]][[2]]
## [1] "sshd" "2461"
##
##
## [[2]]
## [[2]][[1]]
## [1] "CRON"  "21322"
##
## [[2]][[2]]
## [1] "sshd" "4137"
##
##
## [[3]]
## [[3]][[1]]
## [1] "ftpd"  "12297"
##
## [[3]][[2]]
## [1] "kernel" "NA"
##
##
## [[4]]
## [[4]][[1]]
## [1] "UserEventAgent" "43"
##
## [[4]][[2]]
## [1] "sharingd" "30299"
```

```
##
##
## [[5]]
## [[5]][[1]]
## [1] "sshd"  "25537"
##
## [[5]][[2]]
## [1] "sshd"  "24525"
```

```
pid_app <- lapply(pid_app, function(x) do.call(rbind, x))
```

The rest is straight forward. The last part of each line is the message. Thus we shall make data frame of each parsed part and create a finale column containg which file the observation belong to, and then merge them all. We can confirm that the length of the merged data frame has preserved the 99958 rows as we discussed earlier.

```
file_name <- gsub("# |\\.log", "", ll[names])

df_list <- list()
for (i in 1:5) {
    df_list[[i]] <- data.frame(
        `date-time` = dt[[i]],
        `logging host` = log_host[[i]],
        app = pid_app[[i]][,1],
        PID = pid_app[[i]][,2], message = messages[[i]]
    ) %>% mutate(file = file_name[i])
}
df <- do.call(rbind, df_list)
nrow(df)
```

```
## [1] 99958
```

Data Validation and Exploration

Verify that the PIDs are all numbers.

We can indeed confirm that any observation in PID that is not a number is a NA. Thus we can safely transform it into a numerical column.

```
logic <- grepl("[0-9]+", df$PID)
table(df$PID[!logic])
```

```
##
##  NA
## 945
```

```
df$PID <- suppressWarnings(as.numeric(df$PID))
```

How many lines are in each log file?

From the table below we can observe the number of observations in each file.

```
table(df$file)
```

```
##
##                   auth                  auth2 loghub/Linux/Linux_2k
##                  86839                   7121                   1999
##     loghub/Mac/Mac_2k loghub/OpenSSH/SSH_2k
##                   1999                   2000
```

What are the range of date-times for the messages?

Ignoring the dummy year, we can confirm that the total span of the message are from 27th of March to 31st of December. However, we cannot know if this spans over differnet years since that info was lacking.

```
summary(df$date.time)
```

```
##                       Min.                     1st Qu.
## "2023-03-27 13:06:56.0000" "2023-12-02 14:55:36.2500"
##                     Median                       Mean
## "2023-12-07 16:38:04.0000" "2023-11-18 18:27:54.9961"
##                    3rd Qu.                       Max.
## "2023-12-23 02:48:23.7500" "2023-12-31 22:27:48.0000"
```

for each of the different log files in the combined file?

- **auth:** spans from 30th of November to 31st of December
- **auth2:** spans from 27th of March to 20th of April.
- **loghub/Linux/Linux_2k:** spans from 14th of June to 27th of July.
- **loghub/Mac/Mac_2k:** spans from 1st of July to 8th of July.
- **loghub/OpenSSH/SSH_2k:** spans over the same day.

```
df <- df %>%
    mutate_at(vars(file), as.factor)

tapply(df$date.time, df$file, summary)
```

```
## $auth
##                       Min.                     1st Qu.
## "2023-11-30 06:39:00.0000" "2023-12-03 00:52:39.5000"
##                     Median                       Mean
## "2023-12-10 19:20:36.0000" "2023-12-13 07:11:16.7026"
##                    3rd Qu.                       Max.
## "2023-12-23 23:14:49.5000" "2023-12-31 22:27:48.0000"
##
## $auth2
```

```
##                        Min.                      1st Qu.
## "2023-03-27 13:06:56.0000" "2023-03-29 12:49:41.0000"
##                      Median                        Mean
## "2023-03-31 01:35:14.0000" "2023-04-03 11:13:53.6591"
##                     3rd Qu.                        Max.
## "2023-04-06 22:59:21.0000" "2023-04-20 14:14:29.0000"
##
## $`loghub/Linux/Linux_2k`
##                        Min.                      1st Qu.
## "2023-06-14 15:16:01.0000" "2023-06-29 14:44:35.0000"
##                      Median                        Mean
## "2023-07-09 12:16:51.0000" "2023-07-08 10:13:17.1020"
##                     3rd Qu.                        Max.
## "2023-07-17 15:09:16.0000" "2023-07-27 14:42:00.0000"
##
## $`loghub/Mac/Mac_2k`
##                        Min.                      1st Qu.
## "2023-07-01 09:00:55.0000" "2023-07-03 13:48:22.0000"
##                      Median                        Mean
## "2023-07-04 19:42:58.0000" "2023-07-04 21:46:32.3306"
##                     3rd Qu.                        Max.
## "2023-07-06 16:14:05.0000" "2023-07-08 08:10:46.0000"
##
## $`loghub/OpenSSH/SSH_2k`
##                        Min.                      1st Qu.
## "2023-12-10 06:55:46.0000" "2023-12-10 09:12:37.0000"
##                      Median                        Mean
## "2023-12-10 10:14:13.0000" "2023-12-10 09:56:03.4625"
##                     3rd Qu.                        Max.
## "2023-12-10 10:59:43.5000" "2023-12-10 11:04:45.0000"
```

How many days does each log file span?

- **auth:** Time difference of 31.65889 days
- **auth2:** Time difference of 24.04691 days
- **loghub/Linux/Linux_2k:** Time difference of 42.97638 days
- **loghub/Mac/Mac_2k:** Time difference of 6.965174 days
- **loghub/OpenSSH/SSH_2k:** Time difference of 4.149722 hours

```
s <- tapply(df$date.time, df$file, summary)
diffs <- Map(function(x) x[[6]] - x[[1]], s)
```

Do the application names contain numbers? If so, are they just versions, e.g. ssh2, or is there additional structure to the numbers?

Appears to be 11 occurrences of numbers. According to sumo logic (n.d.), an application with the name containing "syslog 1.4.1" utilizes the syslog protocol for logging and operational tasks like audits, monitoring, and troubleshooting. They claim this application follows the standardized syslog architecture, which consists of content, application, and transport layers. On the other hand, as said by Sesek (2016), "BezelServices

255.10" is likely a specific version of the Mac OS X subsystem connecting HID device drivers, preferences, and a UI surface for device feedback. He claims, it supports devices like MagicTrackpad, Bluetooth keyboards, and IR remotes, dynamically selecting them through bundles in a specific directory.

```
table(grepl("[0-9]", df$app))
```

```
##
## FALSE  TRUE
## 99947    11
```

```
df$app[grepl("[0-9]", df$app)]
```

```
##  [1] "syslogd 1.4.1"        "syslogd 1.4.1"        "syslogd 1.4.1"
##  [4] "syslogd 1.4.1"        "syslogd 1.4.1"        "syslogd 1.4.1"
##  [7] "syslogd 1.4.1"        "BezelServices 255.10" "BezelServices 255.10"
## [10] "BezelServices 255.10" "BezelServices 255.10"
```

Is the host value constant/the same for all records in each log file?

It appears as is it is constant in all files except for **loghub/Mac/Mac_2k**.

```
tapply(df$logging.host, df$file, table)
```

```
## $auth
##
## ip-172-31-27-153
##            86839
##
## $auth2
##
## ip-10-77-20-248
##            7121
##
## $`loghub/Linux/Linux_2k`
##
## combo
##  1999
##
## $`loghub/Mac/Mac_2k`
##
##   airbears2-10-142-108-38  airbears2-10-142-110-255         authorMacBook-Pro
##                        15                         79                       554
## calvisitor-10-105-160-179 calvisitor-10-105-160-181 calvisitor-10-105-160-184
##                        19                          6                        39
## calvisitor-10-105-160-205 calvisitor-10-105-160-210  calvisitor-10-105-160-22
##                        30                          9                         7
## calvisitor-10-105-160-226 calvisitor-10-105-160-237  calvisitor-10-105-160-37
##                        17                         53                        12
##  calvisitor-10-105-160-47  calvisitor-10-105-160-85  calvisitor-10-105-160-95
##                         6                         83                       140
## calvisitor-10-105-161-176 calvisitor-10-105-161-225 calvisitor-10-105-161-231
```

```
##                          6                          16                          2
##  calvisitor-10-105-161-77 calvisitor-10-105-162-105 calvisitor-10-105-162-107
##                          2                         338                         13
## calvisitor-10-105-162-108 calvisitor-10-105-162-124 calvisitor-10-105-162-138
##                          4                          28                          3
## calvisitor-10-105-162-175 calvisitor-10-105-162-178 calvisitor-10-105-162-211
##                          4                         256                          3
## calvisitor-10-105-162-228  calvisitor-10-105-162-32  calvisitor-10-105-162-81
##                          5                          27                          3
##  calvisitor-10-105-162-98  calvisitor-10-105-163-10 calvisitor-10-105-163-147
##                          8                          34                          5
## calvisitor-10-105-163-168 calvisitor-10-105-163-202 calvisitor-10-105-163-253
##                          4                         137                         26
##  calvisitor-10-105-163-28   calvisitor-10-105-163-9
##                          2                           4
##
## $`loghub/OpenSSH/SSH_2k`
##
## LabSZ
##  2000
```

What are the most common apps (daemons/programs) that are logging information on each of the different hosts?

The table below displays for each logging host which app hade the most ocurrances, sorted by highest occurrence to lowest, displaying top 10.

```
df %>%
    group_by(logging.host, app) %>%
    count() %>%
    arrange(logging.host, desc(n)) %>%
    ungroup() %>%
    group_by(logging.host) %>%
    slice_max(n, n = 1) %>%
    arrange(desc(n)) %>%
    head(10)
```

```
## # A tibble: 10 x 3
## # Groups:   logging.host [10]
##    logging.host              app        n
##    <chr>                     <chr>  <int>
##  1 ip-172-31-27-153          sshd   85246
##  2 ip-10-77-20-248           sshd    4095
##  3 LabSZ                     sshd    2000
##  4 combo                     ftpd     916
##  5 authorMacBook-Pro         kernel   192
##  6 calvisitor-10-105-162-105 kernel   105
##  7 calvisitor-10-105-162-178 kernel   100
##  8 calvisitor-10-105-160-95  kernel    67
##  9 calvisitor-10-105-163-202 kernel    43
## 10 airbears2-10-142-110-255  kernel    35
```

Logins - valid and invalid

To see a pattern in how a valid message look like, we begin by visuallt observing the different messages. Then we create two columns containing what we think would be classified ass valid, invalid, or overload (too many attempts). Making sure we have no common elements we, check using `intersect()`. We also try to see what type of observations are left by removing any matches to either valid or invalid. To make it easier to identify which type of messages are left we shall also create a list of closing connections since it is acommon phenomena. By iterating through this process, and observing different entries in `truly_left`, we are able to identify three list of possible matches (with no intersections).

```
valids <- matches(df$message,"(new group)|(Accepted)|(opened)|((C|c)onnection from [0-9])|(New session)

invalids <- matches(df$message,"(^Invalid)|(Did not)|(Could not)|(POSSIBLE)|(Failed password)|(input_us

overload <- matches(df$message,"(many)|(maximum authentication)|(ignoring max retries)")

tot_match <- c(valids, invalids, overload)
left <- setdiff(df$message, tot_match)

closed <- matches(left,"(Bye Bye)|(session closed)|(Connection closed)|(Received disconnect)|(Disconnec

truly_left <- setdiff(left, closed)
length(left)
```

```
## [1] 4395
```

```
intersect(valids, invalids)
```

```
## character(0)
```

```
intersect(overload, invalids)
```

```
## character(0)
```

```
intersect(overload, valids)
```

```
## character(0)
```

Find valid/successful logins, what are the user names and Internet Protocol (IP) addresses of those logins?

In the `tester` variable we try different patterns, check how many unique occurrences there are and from there on try to extract user or IP. For instance, a common pattern is "session opened for user" proceeded by a username.

```
head(valids)
```

```
## [1] "pam_unix(cron:session): session opened for user root by (uid=0)"
## [2] "pam_unix(cron:session): session opened for user root by (uid=0)"
```

```
## [3] "pam_unix(cron:session): session opened for user root by (uid=0)"
## [4] "pam_unix(cron:session): session opened for user root by (uid=0)"
## [5] "pam_unix(cron:session): session opened for user root by (uid=0)"
## [6] "pam_unix(cron:session): session opened for user root by (uid=0)"
```

```
tester <- matches(valids, "session opened for user")
names <- str_extract(tester, "(?<=session opened for user )([^ ]+)")
users <- unique(names)
users
```

```
##  [1] "root"           "ubuntu"         "elastic_user_7" "elastic_user_2"
##  [5] "elastic_user_8" "elastic_user_5" "elastic_user_0" "elastic_user_4"
##  [9] "elastic_user_3" "elastic_user_1" "elastic_user_6" "elastic_user_9"
## [13] "cyrus"          "news"           "test"           "fztu"
```

Similarily to before, after we've extracted the matches we remove those lines from the file and try to find new ones. The approach here is that it i probably best to find all different names and see the possible IP addresses, then see if there are sole IP addresses without names. Sampling make it appear as if there are only IP addresses left.

```
any_user <- paste(users, sep="", collapse="|")
all_identified_users <- matches(valids, any_user)
users_left <- setdiff(valids, all_identified_users)
sample(users_left, 5)
```

```
## [1] "connection from 202.82.200.188 () at Fri  2005"
## [2] "connection from 172.181.208.156 () at Tue  2005"
## [3] "connection from 211.42.188.206 () at Fri  2005"
## [4] "connection from 211.72.151.162 () at Wed  2005"
## [5] "connection from 210.245.165.136 () at Wed  2005"
```

We shall begin by extracting The IP addresses of the known users and then the IP addresses from all non declared connections. Here are some possible differnet user ip combinations. At appears as if several users share the same ip adress. But there are no appearances of "cyrus", "news", or "test".

```
ip_rx <- "[0-9]{1,3}\\.[0-9]{1,3}\\.[0-9]{1,3}\\.[0-9]{1,3}"
rx <- paste("(", any_user, ")", " from ", ip_rx, sep ="")
user_with_ip <- matches(all_identified_users, rx)
unique(str_extract(user_with_ip, rx))
```

```
##  [1] "ubuntu from 85.245.107.41"         "elastic_user_7 from 127.0.0.1"
##  [3] "elastic_user_2 from 85.245.107.41" "elastic_user_8 from 85.245.107.41"
##  [5] "elastic_user_5 from 85.245.107.41" "elastic_user_7 from 85.245.107.41"
##  [7] "elastic_user_0 from 85.245.107.41" "elastic_user_4 from 85.245.107.41"
##  [9] "elastic_user_3 from 85.245.107.41" "elastic_user_8 from 24.151.103.17"
## [11] "elastic_user_1 from 85.245.107.41" "elastic_user_0 from 24.151.103.17"
## [13] "elastic_user_6 from 24.151.103.17" "elastic_user_7 from 24.151.103.17"
## [15] "elastic_user_9 from 24.151.103.17" "elastic_user_9 from 85.245.107.41"
## [17] "elastic_user_6 from 85.245.107.41" "elastic_user_1 from 24.151.103.17"
## [19] "elastic_user_3 from 24.151.103.17" "elastic_user_2 from 24.151.103.17"
## [21] "elastic_user_5 from 24.151.103.17" "elastic_user_4 from 24.151.103.17"
## [23] "elastic_user_3 from 95.93.96.191"  "elastic_user_9 from 95.93.96.191"
## [25] "elastic_user_2 from 95.93.96.191"  "ubuntu from 95.93.96.191"
## [27] "fztu from 119.137.62.142"
```

It appears as if they are not related to an ip-adress. Thus it is whats seen above that are the different user ip adress combinations.

```
rx <- "cyrus|news|test"
user_with_ip <- matches(all_identified_users, rx)
unique(user_with_ip)
```

```
## [1] "session opened for user cyrus by (uid=0)"
## [2] "session opened for user news by (uid=0)"
## [3] "session opened for user test by (uid=509)"
```

For all of the ip, we have the following:

```
ips <- matches(valids, ip_rx)
ips_isolated <- str_extract(ips, ip_rx)
valid_ips <- unique(ips_isolated)
valid_ips
```

```
##  [1] "0.0.0.0"         "85.245.107.41"   "186.219.213.14"  "24.151.103.17"
##  [5] "182.32.215.94"   "127.0.0.1"       "95.93.96.191"    "24.54.76.216"
##  [9] "82.252.162.81"   "222.33.90.199"   "210.245.165.136" "218.69.108.57"
## [13] "210.118.170.95"  "211.167.68.59"   "61.74.96.178"    "208.62.55.75"
## [17] "210.223.97.117"  "202.82.200.188"  "203.101.45.59"   "62.99.164.82"
## [21] "63.197.98.106"   "211.72.2.106"    "211.72.151.162"  "221.4.102.93"
## [25] "211.57.88.250"   "81.171.220.226"  "206.196.21.129"  "217.187.83.139"
## [29] "82.83.227.67"    "220.94.205.45"   "211.107.232.1"   "212.65.68.82"
## [33] "83.116.207.11"   "218.146.61.230"  "207.30.238.8"    "82.68.222.194"
## [37] "82.68.222.195"   "216.12.111.241"  "211.42.188.206"  "67.95.49.172"
## [41] "84.102.20.2"     "206.47.209.10"   "217.187.83.50"   "172.181.208.156"
## [45] "218.38.58.3"     "119.137.62.142"
```

Find the invalid user login/ids

What were the associated IPs

The invalid logins are made by 1826 unqiue different ip adresses.

```
ips <- matches(invalids, ip_rx)
ips_isolated <- str_extract(ips, ip_rx)
invalid_ip <- unique(ips_isolated)
length(invalid_ip)
```

```
## [1] 1826
```

```
head(invalid_ip)
```

```
## [1] "187.12.249.74"   "196.200.90.236"  "1.93.26.70"      "122.225.109.208"
## [5] "218.75.153.170"  "124.95.165.186"
```

Were there multiple invalid user logins from the same IP addresses

Below we see the top attempts. One has even tried thousands of times.

```
top_attempts <- data.frame(table(ips_isolated)) %>% arrange(desc(Freq))
head(top_attempts, 10)
```

```
##      ips_isolated Freq
## 1    188.87.35.25 2072
## 2  218.75.153.170  733
## 3  183.62.140.253  582
## 4  92.209.161.222  566
## 5   218.25.17.234  409
## 6    220.99.93.50  409
## 7  61.197.203.243  409
## 8    108.63.28.45  373
## 9    123.57.51.31  360
## 10  177.43.213.35  356
```

Were there valid logins from those IP addresses

There appears to be overlaps. Five of them have invalid attempts.

```
any(valid_ips %in% invalid_ip)
```

```
## [1] TRUE
```

```
intersect(invalid_ip, valid_ips)
```

```
## [1] "85.245.107.41"  "186.219.213.14" "24.151.103.17"  "182.32.215.94"
## [5] "127.0.0.1"
```

Are there multiple IPs using the same invalid login?

First we find all the unique attempts with unique ips. Then, we remove the ips, and if there are more ocurrances of one attempt, it implies different ips has attempted the same login. Appear to be correctly extracted. Below we can observer that indeaed several different IPs use the same login usernames, most commonly "admin".

```
invalid_rx <- paste("Invalid user ([^ ]+) from ", ip_rx, sep="")
unique_invalids <- unique(matches(invalids, invalid_rx))
unique_attempts <- gsub(ip_rx, "", unique_invalids)
sample(unique_attempts, 5)
```

```
## [1] "Invalid user oracle from "  "Invalid user support from "
## [3] "Invalid user pi from "      "Invalid user Sorin from "
## [5] "Invalid user vyatta from "
```

```
res_df <- data.frame(table(unique_attempts)) %>% arrange(desc(Freq))
head(res_df, 10)
```

```
##                unique_attempts Freq
## 1     Invalid user admin from   825
## 2   Invalid user ftpuser from   323
## 3    Invalid user D-Link from   280
## 4     Invalid user debug from   267
## 5       Invalid user log from   260
## 6  Invalid user PlcmSpIp from   252
## 7     Invalid user guest from   230
## 8     Invalid user karaf from   225
## 9    Invalid user vyatta from   223
## 10       Invalid user pi from   219
```

Are these related IPs, e.g., from the same network/domain? Allthoug an IP is from the same network most of the time if only the last digits differ, it does not always hold (Nielsen, 2017). However, due to the scope of the assignment we will make such assumption. We want to take all the unique ocurrances, extract the ip, then compare similariteis to identify wheter they are from the same network or different. Luckily we have extracted the ip from before. We shall remove the last two "octet", count the ocurrances of the same network.

Below we see the most common occurances. From googling, it appears as the three most common ones are from Amazon, 1&1 Internet AG, and China Telecom (Find IP Address, 2023).

```
rx <- "[0-9]{1,3}\\.[0-9]{1,3}"
res <- str_extract(invalid_ip, rx)
counts <- data.frame(table(res)) %>% arrange(desc(Freq))
head(counts, 10)
```

```
##        res Freq
## 1   54.173   92
## 2   87.106   72
## 3  122.225   63
## 4   82.165   19
## 5   85.214   15
## 6   61.174   12
## 7   109.63   11
## 8   87.118   11
## 9   87.230   10
## 10   92.61   10
```

What IP address had too many authentication failures.

Below we have found all unique rows with an overload message, matched which ones that contains an IP address, then extracted all the unique ocurrances.

```
unique(str_extract(matches(unique(overload), ip_rx), ip_rx))
```

```
##   [1] "122.176.37.221"  "95.152.57.58"    "90.144.183.19"   "186.128.152.44"
```

```
##   [5] "201.177.23.130"  "190.178.62.6"    "201.43.243.37"   "68.33.123.70"
##   [9] "190.49.42.132"   "112.135.124.229" "62.73.115.98"    "2.60.103.231"
##  [13] "31.162.29.148"   "179.39.18.38"    "115.209.121.31"  "113.89.184.179"
##  [17] "191.85.133.91"   "111.0.82.180"    "31.162.4.186"    "5.37.195.14"
##  [21] "118.180.18.102"  "5.144.9.33"      "79.165.2.209"    "14.185.87.49"
##  [25] "190.96.200.229"  "5.74.204.136"    "186.133.159.181" "218.60.136.106"
##  [29] "58.19.145.242"   "117.192.138.116" "112.237.146.110" "179.37.3.130"
##  [33] "217.100.114.122" "39.71.33.50"     "115.211.146.244" "221.122.101.203"
##  [37] "190.97.81.95"    "113.124.141.181" "5.141.40.99"     "143.208.24.234"
##  [41] "115.58.179.206"  "201.178.186.182" "123.96.41.232"   "123.115.51.110"
##  [45] "182.32.215.94"   "181.25.206.27"   "106.57.58.19"    "181.23.168.176"
##  [49] "181.25.201.155"  "111.40.168.90"   "122.189.198.238" "60.187.118.40"
##  [53] "122.191.89.89"   "42.184.142.151"  "82.64.2.59"      "114.32.100.101"
##  [57] "122.244.28.82"   "123.153.146.183" "181.26.186.35"   "183.152.79.79"
##  [61] "1.30.211.144"    "5.167.75.191"    "93.120.176.237"  "78.106.21.86"
##  [65] "112.251.168.248" "119.193.140.176" "58.19.144.50"    "122.112.235.133"
##  [69] "122.189.193.214" "49.4.143.105"    "61.166.73.66"    "122.191.88.115"
##  [73] "186.130.83.53"   "123.96.5.168"    "111.40.30.206"   "188.18.252.218"
##  [77] "125.107.136.165" "190.107.182.33"  "112.101.164.200" "223.244.185.76"
##  [81] "182.243.87.6"    "91.243.236.123"  "49.4.143.5"      "222.89.76.13"
##  [85] "191.81.42.216"   "183.93.215.158"  "122.190.143.18"  "61.183.117.250"
##  [89] "123.120.200.51"  "73.231.4.205"    "110.78.174.75"   "49.4.143.181"
##  [93] "186.128.141.232" "46.89.129.145"   "59.174.52.12"    "103.230.120.26"
##  [97] "122.189.197.241" "123.119.111.172" "123.164.142.82"  "175.162.187.121"
## [101] "60.165.208.28"   "218.91.34.237"   "49.84.87.84"     "222.187.86.51"
## [105] "77.231.252.103"  "95.190.198.34"   "94.154.25.149"   "183.146.159.20"
## [109] "126.59.251.31"   "183.93.253.159"  "219.82.145.223"  "122.163.61.218"
## [113] "14.54.210.101"   "105.101.221.33"  "37.110.24.51"    "181.25.189.115"
## [117] "178.219.248.139" "177.38.145.209"  "191.83.152.32"   "170.79.155.119"
## [121] "46.30.160.83"    "201.178.245.106" "181.211.173.182" "122.144.136.83"
## [125] "1.189.205.173"   "181.23.26.185"   "201.27.216.125"  "179.208.151.103"
## [129] "119.193.140.203" "182.243.85.75"   "61.174.116.31"   "111.40.166.130"
## [133] "186.129.147.223" "37.78.105.176"   "58.100.135.31"   "178.161.33.80"
## [137] "179.38.76.250"   "186.47.222.98"   "201.178.81.113"  "68.182.39.76"
## [141] "122.5.240.60"
```

Sudo commands

```
sudos <-
    df %>%
    filter(app == "sudo") %>%
    select(message) %>%
    unique()
```

What are the executables/programs run via sudo

```
sudo_command <- matches(sudos$message, "COMMAND")
rx <- "(?<=COMMAND)([^ ]+)"
unique(str_extract(sudo_command, rx))
```

```
##  [1] "=/usr/bin/curl"                     "=/usr/bin/apt-key"
##  [3] "=/usr/bin/apt-get"                  "=/usr/bin/tee"
##  [5] "=/usr/sbin/update-rc.d"             "=/usr/bin/vim"
##  [7] "=/bin/hostname"                     "=/usr/bin/hostnamectl"
##  [9] "=/usr/sbin/service"                 "=/usr/bin/dpkg"
## [11] "=./filebeat"                        "=/bin/su"
## [13] "=/usr/share/filebeat/bin/filebeat" "=/bin/cp"
## [15] "=/usr/bin/filebeat.sh"              "=/bin/rm"
## [17] "=/bin/chmod"                        "=/bin/mkdir"
## [19] "=/usr/bin/apt"                      "=./create_n_users.sh"
## [21] "=/usr/sbin/groupadd"                "=/sbin/resolvconf"
## [23] "=/usr/bin/hexdump"                  "=/bin/chown"
## [25] "=/usr/bin/vi"                       "=/sbin/auditctl"
## [27] "=/sbin/ausearch"                    "=/usr/bin/tail"
## [29] "=/bin/ls"
```

By what user

```
sudo_command <- matches(sudos$message, "USER=")
rx <- "(?<=USER=)([^ ]+)"
unique(str_extract(sudo_command, rx))
```

```
## [1] "root"
```

What machine are they on?

Linux

Sources

Find IP Address. (2023). IP - Lookup and Locator, Available Online: https://www.findip-address.com/ [Accessed 1 May 2023]

Nielsen, C. L. (2017). Answer to 'Would IP Addresses with the Same First 3 Octets Necessarily Be from the Same Origin?', Server Fault, Available Online: https://serverfault.com/a/828348 [Accessed 1 May 2023]

Pon, H. (2011). Answer to ' "Incomplete Final Line" Warning When Trying to Read a .Csv File into R', Stack Overflow, Available Online: https://stackoverflow.com/a/5996412 [Accessed 28 April 2023]

Sesek, R. (2016). BezelServices on OS X, Available Online: https://robert.sesek.com/2016/3/bezelservices_on_os_x.html [Accessed 29 April 2023]

sumo logic. (n.d.). What Is Syslog?, Available Online: https://www.sumologic.com/syslog/ [Accessed 29 April 2023]