STA141B Project 2

2023-04-27

First, we will begin by reading in the five different tables from the MergedAuth log file. I received help from Piazza to find the best method to do this: using cumsum(). Essentially, we can use the cumulative sum in conjunction with grepl to find each occurence of the table title pattern. Grepl returns true and false values for each index where the pattern matches and cumsum uses this to denote the start of a new table.

The pattern we use to match each line, "^# [^]+", can be broken down. "^" denotes that the pattern must match at the start of the line. "#" is the literal character, as each title begins with it. [ ] indicates a set of characters, but the "^" is a negation, not a starting signal. This means that we want to match for any characters that are not a space. Lastly, the + indicates that we want to match the set at least once. I got this from the rx code written in the readTableApproach.R file.

Otherwise, # luckily only occurs mid-line in the file, and we get exactly five matches.

Split then uses the location of cumsum increases to create a new table. Lastly, we remove the first entry as it is a white space before the first match is located.

We have to make some changes to our regex expression here. Now, some of the app names have a dash in between. Therefore, instead of allowing only letters, we take any type of character. In addition, we need to find a way to process a colon in the case that it does not have the bracket expression and the colon arrives immediately after sudo.

```
setwd('C:/Users/HP/Documents/STA 141B/STA141B_Project2/')
ll = readLines('MergedAuth.log')
```

```
## Warning in readLines("MergedAuth.log"): incomplete final line found on
## 'MergedAuth.log'
```

```
starts = cumsum(grepl("^# [^ ]+", ll))
table_list = split(ll, starts)

table_list = table_list[-1]
length(table_list)
```

```
## [1] 5
```

```
str(table_list)
```

```
## List of 5
##  $ 1: chr [1:86841] "# auth.log" "Nov 30 06:39:00 ip-172-31-27-153 CRON[21882]: pam_unix(cron:sessio
##  $ 2: chr [1:7123] "# auth2.log" "Mar 27 13:06:56 ip-10-77-20-248 sshd[1291]: Server listening on 0.
##  $ 3: chr [1:2001] "# loghub/Linux/Linux_2k.log" "Jun 14 15:16:01 combo sshd(pam_unix)[19939]: authe
##  $ 4: chr [1:2001] "# loghub/Mac/Mac_2k.log" "Jul  1 09:00:55 calvisitor-10-105-160-95 kernel[0]: IO
##  $ 5: chr [1:2001] "# loghub/OpenSSH/SSH_2k.log" "Dec 10 06:55:46 LabSZ sshd[24200]: reverse mapping
```

```r
# Now, we split up each table.

auth.log = table_list[1]
auth2.log = table_list[2]
linux_2k.log = table_list[3]
mac_2k.log = table_list[4]
ssh_2k.log = table_list[5]

#Next, we convert it from lists into dataframes. Our full string is listed under
#one variable, and it is now our job to begin to break it up into multiple
#variables.
source("getCaptures.R")

#Collaborated with Michael Karim

#This pattern went through many alterations. This is the final version as it is
#used to read in all of the tables. The primary alterations that I can think of
#off the top of my head:

#Adding an optional whitespace to the datetime to allow for extra spacing on
#some tables

#Creating the conditional for reading in the IP address. This was necessary for
#the third table, where the logging host is simply "combo". It does not take in
#any numbers, so it has to be recorded as words only.

#Initially, the app name took in a string. However, with the addition of dashes
#and other various characters, I simply let it take anything. To be completely
#honest, I do not know how this question mark works; I simply tested around with
#it as I needed to be able to reach the ":" when the PID did not exist
#(when app = sudo).

# The bracketed PID underwent the most editing. Because of the addition of sudo
#in auth2.log, many lines do not have a PID # at all. The ?: at the start and
#the ? at the end make this section optional.

#The last segment simply reads in all possible characters after the ":".

#For clarity without messing up rx expression:
#rx_test = "^([a-zA-Z]+ \\s*[0-9]+ [0-9]{2}:[0-9]{2}:[0-9]{2}) ([\\w-]+|[a-zA-Z]
#+-[0-9-]+) (.*?)(?:\\[([0-9]+)\\])?:* (.*)"

rx_test = "^([a-zA-Z]+ \\s*[0-9]+ [0-9]{2}:[0-9]{2}:[0-9]{2}) ([\\w-]+|[a-zA-Z]+-[0-9-]+) (.*?)(?:\\[([

data_prep <- function(file) {
  file = as.data.frame(file)
  file = file[-1,]
  file = file[-length(file)]
  file
}

table_test_function <- function(rx, file) {
  w = grepl(rx, file, perl = TRUE)
```

```
  print(table(w))
  m = gregexpr(rx, file, perl = TRUE)
  print(table(sapply(m, length)))
  print(table(sapply(m, `[`, 1) == 1))
  s = attr(m[[1]], "capture.start")
  substring(file[1], s, s + attr(m[[1]], "capture.length"))
}

table_create <- function(rx, file) {
  caps = getCaptures(rx, file, row.names = NULL)
  dim(caps)
  var_names = c("date-time", "logging host", "app", "PID", "message")
  colnames(caps) = var_names
  file_name <- rep(deparse(substitute(file)), nrow(caps))
#Suggested by ChatGPT to get the name of the file passed in, rather than accessing the file itself
  caps <- cbind(caps, file_name)
  caps
}

#For running the file in final iteration

auth.log = data_prep(table_list[1])
table_test_function(rx_test, auth.log)
```

```
## w
##  TRUE
## 86839
##
##     1
## 86839
##
##  TRUE
## 86839
```

```
## [1] "Nov 30 06:39:00 "
## [2] "ip-172-31-27-153 "
## [3] "CRON["
## [4] "21882]"
## [5] "pam_unix(cron:session): session closed for user root"
```

```
auth.log = table_create(rx_test, auth.log)


auth2.log = data_prep(table_list[2])
table_test_function(rx_test, auth2.log)
```

```
## w
## TRUE
## 7121
##
##    1
## 7121
```

```
##
## TRUE
## 7121

## [1] "Mar 27 13:06:56 "
## [2] "ip-10-77-20-248 "
## [3] "sshd["
## [4] "1291]"
## [5] "Server listening on 0.0.0.0 port 22."
```

```
auth2.log = table_create(rx_test, auth2.log)

linux_2k.log = data_prep(table_list[3])
table_test_function(rx_test, linux_2k.log)
```

```
## w
## TRUE
## 1999
##
##    1
## 1999
##
## TRUE
## 1999

## [1] "Jun 14 15:16:01 "
## [2] "combo "
## [3] "sshd(pam_unix)["
## [4] "19939]"
## [5] "authentication failure; logname= uid=0 euid=0 tty=NODEVssh ruser= rhost=218.188.2.4 "
```

```
linux_2k.log = table_create(rx_test, linux_2k.log)

mac_2k.log = data_prep(table_list[4])
table_test_function(rx_test, mac_2k.log)
```

```
## w
## TRUE
## 1999
##
##    1
## 1999
##
## TRUE
## 1999

## [1] "Jul  1 09:00:55 "
## [2] "calvisitor-10-105-160-95 "
## [3] "kernel["
## [4] "0]"
## [5] "IOThunderboltSwitch<0>(0x0)::listenerCallback - Thunderbolt HPD packet for route = 0x0 port = 1
```

```
mac_2k.log = table_create(rx_test, mac_2k.log)

ssh_2k.log = data_prep(table_list[5])
table_test_function(rx_test, ssh_2k.log)
```

```
## w
## TRUE
## 1999
##
##    1
## 1999
##
## TRUE
## 1999
```

```
## [1] "Dec 10 06:55:46 "
## [2] "LabSZ "
## [3] "sshd["
## [4] "24200]"
## [5] "reverse mapping checking getaddrinfo for ns.marryaldkfaczcz.com [173.234.31.186] failed - POSSI
```

```
ssh_2k.log = table_create(rx_test, ssh_2k.log)
```

```
merged_auth = rbind(auth.log, auth2.log, linux_2k.log, mac_2k.log, ssh_2k.log)
```

Data Validation and Exploration

The first thing that we want to check is for lines where there is no app or PID provided so that we can set their corresponding value to NA.

```
missing_values <- function(file) {
  missing_values <- is.na(file$app)
  print(sum(missing_values))
  is.na(file$PID) <- file$PID == ""
  missing_values <- is.na(file$PID)
  print(sum(missing_values))
  return(file)
}

merged_auth <- missing_values(merged_auth)
```

```
## [1] 0
## [1] 1024
```

Convert PID to integer

```
merged_auth$PID = as.integer(merged_auth$PID)
```

Convert date-time to POSIXct

```
merged_auth$`date-time` = as.POSIXct(strptime(merged_auth$`date-time`, "%b %d %H:%M:%S"))

table(is.na(merged_auth$`date-time`))
```

```
##
## FALSE
## 99957
```

Great. We did not create any NA values during coercion.

This coerced in the year 2023 into our datetime as it is needed for a POSIXct datatype. As there is no date provided to us, we will just proceed forward using this value.

How many lines are in each log file? Now is a great time to split apart our merged_auth file again based on their file_name. We need to do this for individual computations like the range of dates for each log-file and log-file specific valid/invalid log-in attempts.

```
auth.log = merged_auth %>% filter(file_name == "auth.log") %>%
  select(`date-time`, `logging host`, app, PID, message)

auth2.log = merged_auth %>% filter(file_name == "auth2.log") %>%
  select(`date-time`, `logging host`, app, PID, message)
linux_2k.log = merged_auth %>% filter(file_name == "linux_2k.log") %>%
  select(`date-time`, `logging host`, app, PID, message)

mac_2k.log = merged_auth %>% filter(file_name == "mac_2k.log") %>%
  select(`date-time`, `logging host`, app, PID, message)

ssh_2k.log = merged_auth %>% filter(file_name == "ssh_2k.log") %>%
  select(`date-time`, `logging host`, app, PID, message)

dim(auth.log)
```

```
## [1] 86839     5
```

```
dim(auth2.log)
```

```
## [1] 7121    5
```

```
dim(linux_2k.log)
```

```
## [1] 1999    5
```

```
dim(mac_2k.log)
```

```
## [1] 1999    5
```

```
dim(ssh_2k.log)
```

```
## [1] 1999    5
```

After we removed the empty lines and the filename line, auth.log has 86,839 total lines. That's hefty! Auth2.log has the second most lines with 7121 rows. Linux_2k.log, mac_2k.log, and ssh_2k.log all have 1999 lines each.

What are the range of date-times for the messages? For each of the different log files in the combined file? How many days does each log file span?

An interesting (perhaps obvious) quirk about log files is that they appear in chronological order. Therefore, we can simply look at the date-time of the last message and subtract it by the first message to get the total time elapsed in between log processing. POSIXct does this handling for us!

```
date_range <- function(file) {
last_line = tail(file, n= 1)
first_line = head(file, n= 1)
last_line$`date-time`
first_line$`date-time`
last_line$`date-time` - first_line$`date-time`
}

date_range(auth.log)
```

```
## Time difference of 31.65889 days
```

```
date_range(auth2.log)
```

```
## Time difference of 24.04691 days
```

```
date_range(linux_2k.log)
```

```
## Time difference of 42.97638 days
```

```
date_range(mac_2k.log)
```

```
## Time difference of 6.965174 days
```

```
date_range(ssh_2k.log)
```

```
## Time difference of 4.149167 hours
```

From this, we see that the first date in auth.log is November 30th, and the last date is December 31st. By subtracting the two values, we see that there is an overall time difference of 31.65889 days.

We repeat this process for each file. For auth2.log, we get a difference of 24.0469 days. Linux_2k.log = 42.976 mac_2k.log = 6.965 Astoundingly, the ssh_2k.log only spans 4.149 hours! That speaks to how many operations occur with a server in a short amount of time.

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

First, we look at how many unique names there are.

```
head(unique(auth.log$app))
```

```
## [1] "CRON" "sshd"
```

```
unique(auth2.log$app)
```

```
## [1] "sshd"           "systemd-logind" "systemd"        "sudo"
## [5] "CRON"           "su"             "chpasswd"       "useradd"
## [9] "groupadd"
```

```
head(unique(linux_2k.log$app))
```

```
## [1] "sshd(pam_unix)" "su(pam_unix)"   "logrotate"      "ftpd"
## [5] "cups"           "syslogd"
```

```
length(unique(linux_2k.log$app))
```

```
## [1] 30
```

```
head(unique(mac_2k.log$app))
```

```
## [1] "kernel"                "com.apple.CDScheduler" "QQ"
## [4] "mDNSResponder"         "symptomsd"             "configd"
```

```
length(unique(mac_2k.log$app))
```

```
## [1] 66
```

```
unique(ssh_2k.log$app)
```

```
## [1] "sshd"
```

We clearly see that in the auth.log file, there are only two types of application names. This will change in the other files. Auth2 has 9 different application names, but none of them have numbers involved. Linux_2k is the first to introduce lines with no app names. It also balloons up the total application name total to 30.

Mac_2K takes this a step further with 66 total application names. Clearly, the internal processes of this Mac server are much more complex than the first two files.

Lastly, ssh_2k.log returns to the most simple form of only having sshd processes.

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

```
length(unique(auth.log$`logging host`))
```

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

```
length(unique(auth2.log$`logging host`))
```

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

```
length(unique(linux_2k.log$`logging host`))
```

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

```
length(unique(mac_2k.log$`logging host`))
```

```
## [1] 38
```

```
length(unique(ssh_2k.log$`logging host`))
```

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

```
head(unique(mac_2k.log$`logging host`),n = 10)
```

```
##  [1] "calvisitor-10-105-160-95"  "authorMacBook-Pro"
##  [3] "calvisitor-10-105-163-202" "calvisitor-10-105-160-237"
##  [5] "calvisitor-10-105-160-184" "calvisitor-10-105-162-32"
##  [7] "calvisitor-10-105-161-225" "airbears2-10-142-110-255"
##  [9] "calvisitor-10-105-162-105" "calvisitor-10-105-163-10"
```

Here, we have a constant logging host value for every log file, except for Mac. Mac has 38 different logging hosts! For the most part, this is calvisitor or airbears users with different IPs. I assume this is a webserver that local Cal Berkeley students were able to connect to frequently.

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

```
sort(table(auth.log$app), decreasing = TRUE)
```

```
##
##  sshd  CRON
## 85246  1593
```

```
head(sort(table(auth2.log$app), decreasing = TRUE))
```

```
##
##          sshd          CRON          sudo systemd-logind       chpasswd
##          4095          1264           557           452           417
##       systemd
##           238
```

```
head(sort(table(linux_2k.log$app), decreasing = TRUE))
```

```
##
##           ftpd sshd(pam_unix)   su(pam_unix)         kernel        klogind
##            916             677            172             75             46
##      logrotate
##             43
```

```
head(sort(table(mac_2k.log$app), decreasing = TRUE))
```

```
##
##                     kernel               com.apple.cts
##                        774                         166
##                corecaptured                          QQ
##                        158                          75
##                   Microsoft com.apple.WebKit.WebContent
##                          72                          60
```

```
head(sort(table(ssh_2k.log$app), decreasing = TRUE))
```

```
## sshd
## 1999
```

For auth.file: sshd is by far the most commonly used app with 85x more appearances than CRON.

According to Google, the cron command-line utility is a job scheduler on Unix-like operating systems. It is done for resource management on the OS level. sshd is an OpenSSH server process. This is most likely for any server maintenance or operations for one of the professor's webpages.

Auth2.log sees sshd having the lion share, with 4x more than the next most popular. Linux_2k.log sees ftpd having the most, closely followed by sshd(pam_unix). Mac_2k.log mostly has kernel commands by a 7x margin over the second most populous. Interestingly, despite the large variety of apps that it runs, it is not very evenly distributed. ssh_2k.log again only runs sshd apps.

Logins - valid and invalid

We want to test around with a few keywords, like error, failed, or invalid. Invalid often begins at the start of strings but also will appear in the middle, so we should check for both cases.

```
invalid = grep("Invalid ", auth.log$message)
head(auth.log$message[invalid], n = 1)
```

Auth.log1

```
## [1] "Invalid user admin from 187.12.249.74"
```

```
failed = grep(" failed ", auth.log$message)
head(auth.log$message[failed], n = 1)
```

```
## [1] "reverse mapping checking getaddrinfo for 118.11.26.218.internet.sx.cn [218.26.11.118] failed - I
```

```
error = grep("error", auth.log$message)
head(auth.log$message[error], n = 1)
```

```
## [1] "Address 123.249.19.59 maps to error-cdnzz-com.cdnzz.net, but this does not map back to the addr
```

```
idx = grep(" invalid ", auth.log$message)
head(auth.log$message[idx], n = 1)
```

```
## [1] "input_userauth_request: invalid user admin [preauth]"
```

```
length(invalid) + length(error) + length(failed) + length(idx)
```

```
## [1] 29727
```

```
Accepted = grep("Accepted ", auth.log$message)
length(Accepted)
```

```
## [1] 0
```

```
new = grep("New ", auth.log$message)
length(new)
```

```
## [1] 0
```

Overall, there are 29,727 errors/failures. Any message marked as "fail" comes from a message with format "reverse mapping checking getaddrinfo for 118.11.26.218.internet.sx.cn [218.26.11.118] failed - POSSIBLE BREAK-IN ATTEMPT!".

When checking for key phrases New or Accepted, we can see that no outside users were able to join the server. The only person able to access the server is the root-user. This is shown by all of the CRON messages. Given that our logging host only has one IP, the only person capable of accessing the server is 172.31.27.153.

```
cron = auth.log$message[(auth.log$app == "CRON")]
head(auth.log$`logging host`, n = 1)
```

```
## [1] "ip-172-31-27-153"
```

```
sudo = linux_2k.log[linux_2k.log$app == "sudo",]
```

There are equal number of "Invalid" and " invalid " strings. The first Invalid string marks the IP address that attempts to connect ("Invalid user admin from 187.12.249.74"), and the second invalid string describes the input_userauth_request failure ("input_userauth_request: invalid user admin [preauth]").

We will look only at the first Invalid batch and extract the IP addresses from the string.

```
library(stringr)
w = grepl("^Invalid.*([0-9.]+)$", auth.log$message)
table(w)
```

```
## w
## FALSE  TRUE
## 74589 12250
```

```
#This matches the length of the invalid column before.
bad_login = auth.log$message[w]
length(bad_login)
```

```
## [1] 12250
```

```
head(bad_login)
```

```
## [1] "Invalid user admin from 187.12.249.74"
## [2] "Invalid user admin from 122.225.109.208"
## [3] "Invalid user admin from 124.205.250.51"
## [4] "Invalid user guest from 124.205.250.51"
## [5] "Invalid user support from 124.205.250.51"
## [6] "Invalid user avconroot from 218.26.11.118"
```

```
pattern="([0-9.]+)$"
ips =str_extract(bad_login,pattern)

head(unique(ips))
```

```
## [1] "187.12.249.74"   "122.225.109.208" "124.205.250.51"  "218.26.11.118"
## [5] "187.76.80.202"   "221.208.245.210"
```

```
length(unique(ips))
```

```
## [1] 1260
```

```
ip_table = sort(table(ips), decreasing = TRUE)

head(ip_table, n = 20)
```

```
## ips
##    188.87.35.25   218.25.17.234    220.99.93.50  61.197.203.243    123.57.51.31
##             409             409             409             409             360
##  222.161.209.92   173.192.158.3    67.205.20.23   78.129.223.28   222.219.187.9
##             356             303             269             269             146
##   91.135.226.34    108.63.28.45   218.26.11.118   221.179.89.90 120.198.156.138
##             144             141             107             100              92
##   61.235.147.19   198.143.167.2   74.62.217.226    85.95.252.23  87.106.242.123
##              89              64              63              63              63
```

Clearly, we see a lot of repeated attempts from IPs to unsuccessfully login to the server. In the most popularly used IP, we see a wide variety of different usernames that try to login.

```
w = grepl("188.87.35.25", bad_login)
table(w)
```

```
## w
## FALSE  TRUE
## 11841   409
```

```
user_extract = bad_login[w]


pattern="^Invalid user ([a-zA-Z]+)"
user_name =str_extract(user_extract,pattern)
head(user_name)
```

```
## [1] "Invalid user zhangyan" "Invalid user dff"      "Invalid user oracle"
## [4] "Invalid user test"     "Invalid user oracle"   "Invalid user git"
```

```
pattern="([a-zA-Z]+)$"
user_name_final =str_extract(user_name,pattern)
head(user_name_final)
```

```
## [1] "zhangyan" "dff"      "oracle"   "test"     "oracle"   "git"
```

```
sort(table(user_name_final), decreasing = TRUE)
```

```
## user_name_final
##        test      nagios      zabbix       guest        zxin      apache
##          96          65          46          45          20          18
##     zhaowei      tomcat         web    weblogic       cacti         ftp
##          12           9           8           8           7           7
##       squid        Test    webadmin       httpd        java       jboss
##           7           6           6           5           4           4
##           r    sysadmin       nginx      wangyi      oracle        bash
##           4           4           3           3           2           1
##        boot   cactiuser         dff        ftpd         git  guestadmin
##           1           1           1           1           1           1
##   guestuser      guestx    httpdocs     javaprg nagiosadmin  nagiosuser
##           1           1           1           1           1           1
##     nologin       resin      system     webmail    zhangyan
##           1           1           1           1           1
```

```
length(unique(user_name_final))
```

```
## [1] 42
```

In the most popularly used IP, they attempt to login with 42 different usernames. Many of these usernames are also repeated on different IPs. Usernames like test, guest, and sysadmin frequently pop up for each table.

Auth2.log I went about this file a different, and much more arduous way. This was the revised method I came up with, but it is still very inefficient. As a result, my analysis for the other files will be greatly reduced (as I simply do not have the time for it.)

```
invalid = grep("Invalid ", auth2.log$message)
length(invalid)
```

```
## [1] 178
```

```
head(auth2.log$message[invalid], n = 1)
```

```
## [1] "Invalid user support from 95.152.57.58"
```

```
failed = grep("Failed ", auth2.log$message)
df_failed <- as.data.frame(auth2.log$message[failed], col.names = "X1")
length(failed)
```

```
## [1] 717
```

```
head(auth2.log$message[failed], n = 1)
```

```
## [1] "Failed password for elastic_user_7 from 127.0.0.1 port 52942 ssh2"
```

```
error = grep("error", auth2.log$message)
length(error)
```

```
## [1] 189
```

```
head(auth2.log$message[error], n = 1)
```

```
## [1] "error: maximum authentication attempts exceeded for root from 122.176.37.221 port 37107 ssh2 [p
```

```
#Previously, we just tested for invalid inside a string. This no longer gives us unique values, however

idx = grep("input_userauth_request", auth2.log$message)
length(idx)
```

```
## [1] 178
```

```
head(auth2.log$message[idx], n = 1)
```

```
## [1] "input_userauth_request: invalid user support [preauth]"
```

```
df_idx <- as.data.frame(auth2.log$message[idx])
df_error <- as.data.frame(auth2.log$message[error])
df_invalid <- as.data.frame(auth2.log$message[invalid])
df_failed <- as.data.frame(auth2.log$message[failed])

names(df_idx) <- "X1"
names(df_error) <- "X1"
names(df_invalid) <- "X1"
names(df_failed) <- "X1"

#VALID Logins

Accepted = grep("Accepted ", auth2.log$message)
length(Accepted)
```

```
## [1] 226
```

```
head(auth2.log$message[Accepted], n = 1)
```

```
## [1] "Accepted publickey for ubuntu from 85.245.107.41 port 54259 ssh2: RSA SHA256:Kl8kPGZrTiz7g4FO1hy
```

```
new = grep("New ", auth2.log$message)
length(new)
```

```
## [1] 227
```

```
head(auth2.log$message[new], n = 1)
```

```
## [1] "New seat seat0."
```

```
connection_from = grep("Connection from", auth2.log$message)
df_con_from <- as.data.frame(auth2.log$message[connection_from])

length(connection_from)
```

```
## [1] 17
```

```
head(auth2.log$message[connection_from], n = 1)
```

```
## [1] "Connection from 85.245.107.41 port 61663 on 10.77.20.248 port 222"
```

```
valid = length(Accepted) + length(connection_from) + length(new)

#does not work, return to later

df_con_from <- as.data.frame(auth2.log$message[connection_from])
df_Accepted <- as.data.frame(auth2.log$message[Accepted])
df_new <- as.data.frame(auth2.log$message[new])

names(df_new) <- "X1"
names(df_Accepted) <- "X1"
names(df_con_from) <- "X1"
```

Overall, there are 1,262 invalid logins, demarcated by Invalid, Failed, or error. Any message marked as "fail" comes from a message with format "Failed publickey for elastic_user_8". Any message marked as "Invalid" has a message similar to Invalid user support from 95.152.57.58" or "input_userauth_request: invalid user support [preauth]". For errors, the message reads "error: maximum authentication attempts exceeded for root from".

For valid logins, we check three key phrases: Accepted, New, and "Connection from". In total, we get 470 valid logins. Some example messages are: "New session 1 of user ubuntu", "Connection from 85.245.107.41 port 61663 on 10.77.20.248 port 222", and "Accepted publickey for ubuntu from 85.245.107.41 port 54259 ssh2: RSA SHA256:Kl8kPGZrTiz7g4FO1hyqHdsSBBb5Fge6NWOobN03XJg".

For a singular user, it is possible for them to receive multiple of these messages. We will now get the usernames and IPs that have valid logins. We previously combined all of the valid results, but we want to look at each individual table and run their specific regex to pull out all of the valid IPs and usernames.

Let's start first with the "New session" messages to get the usernames.

```
#Exact regex pattern for lines in "New session" dataset
username <- sub(".*\\b(user|seat)\\s+(\\w+)\\b.*", "\\2", df_new$X1)
username = as.data.frame(username)

new_user_table <- username %>% group_by(username) %>% summarise(count = n()) %>% arrange(desc(count))
head(new_user_table)
```

```
## # A tibble: 6 x 2
##   username       count
##   <chr>          <int>
## 1 ubuntu            36
## 2 elastic_user_0    29
## 3 elastic_user_1    25
## 4 elastic_user_6    24
## 5 elastic_user_8    20
## 6 elastic_user_9    20
```

```
length(unique(new_user_table$username))
```

```
## [1] 12
```

Here, we see that there are 12 unique usernames used for creating a new session.

```
pattern="([0-9]+\\.){3}[0-9]+"
ips =str_extract(df_Accepted$X1,pattern)
ips = as.data.frame(ips)

accepted_ips_table <- ips %>% group_by(ips) %>% summarise(count = n()) %>% arrange(desc(count))
head(accepted_ips_table)
```

```
## # A tibble: 4 x 2
##   ips           count
##   <chr>         <int>
## 1 85.245.107.41   174
## 2 24.151.103.17    47
## 3 95.93.96.191      4
## 4 127.0.0.1         1
```

```
length(unique(accepted_ips_table$ips))
```

```
## [1] 4
```

Interestingly, looks like only four total IPs were able to login without error. 85.245.107.41 connected an eyebrow-raising 174 times. 24.151.103.17 logged in 47 times, and the last two landed a meager 5 times combined.

```
pattern="([0-9]+\\.){3}[0-9]+"
ips =str_extract(df_con_from$X1,pattern)
ips = as.data.frame(ips)


con_ips_table <- ips %>% group_by(ips) %>% summarise(count = n()) %>% arrange(desc(count))
head(con_ips_table)
```

```
## # A tibble: 5 x 2
##   ips            count
##   <chr>          <int>
## 1 85.245.107.41     12
## 2 127.0.0.1          2
## 3 182.32.215.94      1
## 4 186.219.213.14     1
## 5 24.151.103.17      1
```

```
length(unique(con_ips_table$ips))
```

```
## [1] 5
```

We found a total of 5 unique IPs in this data, and 3 new IPs, therefore bringing the total up to 7. Now, we will look at invalid attempts.

input_userauth_request: invalid users

```
username_idx <- sub(".*\\buser\\s+(\\w+| ).*", "\\1", df_idx$X1)
username_idx = as.data.frame(username_idx)


idx_user_table <- username_idx %>% group_by(username_idx) %>% summarise(count = n()) %>% arrange(desc(c
head(idx_user_table)
```

```
## # A tibble: 6 x 2
##   username  count
##   <chr>     <int>
## 1 "admin"      45
## 2 " "          43
## 3 "test"        8
## 4 "ubnt"        8
## 5 "pi"          6
## 6 "support"     5
```

```
length(unique(idx_user_table$username))
```

```
## [1] 41
```

41 different usernames attempted and failed to login. Admin and " " (accounts without usernames) held the large majority of these attempts.

errors: maximum authentication attempts

```
pattern="([0-9]+\\.){3}[0-9]+"
ips =str_extract(df_error$X1,pattern)
ips = as.data.frame(ips)


error_ips_table <- ips %>% group_by(ips) %>% summarise(count = n()) %>% arrange(desc(count))
head(error_ips_table)
```

```
## # A tibble: 6 x 2
##   ips             count
##   <chr>           <int>
## 1 49.4.143.105       20
## 2 201.178.81.113      6
## 3 122.163.61.218      5
## 4 181.26.186.35       5
## 5 186.133.159.181     5
## 6 190.178.62.6        5
```

```
length(unique(error_ips_table$ips))
```

```
## [1] 141
```

141 different IPs attempted to connect and exceeded the maximum authentication attempts. Some IPs, like 49.4.143.105, exceeded this amount 20 times!

Failed passwords/publickeys

```
pattern="([0-9]+\\.){3}[0-9]+"
ips =str_extract(df_failed$X1,pattern)
ips = as.data.frame(ips)

failed_ips_table <- ips %>% group_by(ips) %>% summarise(count = n()) %>% arrange(desc(count))
head(failed_ips_table)
```

```
## # A tibble: 6 x 2
##   ips            count
##   <chr>          <int>
## 1 24.151.103.17    157
## 2 34.204.227.175    43
## 3 49.4.143.105      40
## 4 201.178.81.113    32
## 5 122.163.61.218    30
## 6 181.26.186.35     27
```

```
length(unique(failed_ips_table$ips))
```

```
## [1] 105
```

105 different IPs attempted and failed to connect to the server. 24.151.103.17 had 157 different failures to access the server. Perhaps the most interesting thing to us is that 85.245.107.41, the IP with the most successful connections, racked up 15 failures here.

Invalid logins

I think this data will be the most interesting because we are retrieving both username and IP.

```
invalid = grep("Invalid ", auth2.log$message)
length(invalid)
```

```
## [1] 178
```

```
head(auth2.log$message[invalid], n = 1)
```

```
## [1] "Invalid user support from 95.152.57.58"
```

```
df_invalid <- as.data.frame(auth2.log$message[invalid])
names(df_invalid) <- "X1"

pattern="^Invalid user (\\w*|\\s+) from (([0-9]+\\.){3}[0-9]+)"

w = grepl(pattern, df_invalid$X1, perl = TRUE)
table(w)
```

```
## w
## TRUE
##  178
```

```
m = gregexpr(pattern, df_invalid$X1, perl = TRUE)

invalid = getCaptures(pattern, df_invalid$X1, row.names = NULL)
invalid = invalid[-3]

users_table_extra <- invalid %>% group_by(V1) %>% summarise(No_of_Different_IPs = n_distinct(V2), Total_

#The most_popular_username code was supplied by ChatGPT. I could not get slice_max to work in order to

ips_table_extra <- invalid %>% group_by(V2) %>% summarise(No_of_Distinct_Users = n_distinct(V1), Total_A

inv_user_table <- invalid %>% group_by(V1) %>% summarise(count = n()) %>% arrange(desc(count)) %>% renam

inv_ips_table <- invalid %>% group_by(V2) %>% summarise(count = n()) %>% arrange(desc(count)) %>% rename


head(ips_table_extra)
```

```
## # A tibble: 6 x 5
##   IPs             No_of_Distinct_Users Total_Appearances most_popular_~1 login~2
##   <chr>                          <int>             <int> <chr>              <int>
## 1 91.197.232.109                    24                58 admin                 10
## 2 181.26.186.35                      3                 4 test                   2
## 3 122.163.61.218                     2                 3 admin                  2
## 4 181.25.206.27                      2                 2 cloud                  1
## 5 186.133.159.181                    2                 2 admin1                 1
## 6 201.178.81.113                     2                 2 admin                  1
## # ... with abbreviated variable names 1: most_popular_username,
## #   2: login_attempt_count
```

```
head(users_table_extra)
```

```
## # A tibble: 6 x 3
##   Usernames No_of_Different_IPs Total_Appearances
##   <chr>                   <int>             <int>
```

```
## 1 admin                      35                45
## 2 ubnt                        7                 8
## 3 test                        6                 8
## 4 pi                          5                 6
## 5 support                     4                 5
## 6 default                     3                 4
```

```
length(unique(invalid$V1))
```

```
## [1] 41
```

```
length(unique(invalid$V2))
```

```
## [1] 68
```

There are 41 different usernames and 68 different IP addresses. Admin appeared a total of 45 times on 35 different IPs. Once we venture into the IPs table, we see that 91.197.232.109 was by far the most popular, with 24 different users and 58 total appearances. The admin user on this IP attempted to login 10 different times.

Now, we will combine all of the valid/invalid IP and user data into two separate tables to see how they truly breakdown. I originally went with left_join, but it did not work as I hoped given that some IPs were not located in the initial dataset.

```
ips_list <- accepted_ips_table %>% full_join(con_ips_table, by = "ips")  %>% full_join(error_ips_table,

names(ips_list) = c("IP Address","Accepted", "Connected", "Error", "Failed", "Invalid")
ips_list <- cbind(ips_list, Total = rowSums(ips_list[,2:6]))
ips_list <- ips_list %>% arrange(desc(Total))
head(ips_list)
```

```
##       IP Address Accepted Connected Error Failed Invalid Total
## 1  24.151.103.17       47         1     0    157       0   205
## 2  85.245.107.41      174        12     0     15       2   203
## 3 34.204.227.175        0         0     0     43      43    86
## 4   49.4.143.105        0         0    20     40       0    60
## 5 91.197.232.109        0         0     0      0      58    58
## 6 201.178.81.113        0         0     6     32       2    40
```

```
var_list <- new_user_table %>% full_join(idx_user_table, by = "username")  %>% full_join(inv_user_table

names(var_list) = c("Username","New", "IDX", "Invalid")
var_list <- cbind(var_list, Total = rowSums(var_list[,2:4]))
var_list <- var_list %>% arrange(desc(Total))
head(var_list)
```

```
##         Username New IDX Invalid Total
## 1          admin   0  45      45    90
## 2                  0  43       0    43
## 3                  0   0      43    43
## 4         ubuntu  36   0       0    36
## 5 elastic_user_0  29   0       0    29
## 6 elastic_user_1  25   0       0    25
```

For the most part, this checks out. I, however, failed to remember that IDX and Invalid will appear the same amount of times. They are technically both invalid login messages, but are presented in different formats. Interestingly, for blank accounts, it does not know to combine them together. Regardless, for the future, we will not scan for the lowercase invalid messages.

Having all of our data in one place for the IPs is great. While 85.245.107.41 has by far the most accepted values, it actually has less total pop-ups in the log. This is because 24.151.103.17 has 47 acceptances, but 157 failures! These two IPs have by far the most content.

This process was incredibly lengthy. Since I delved into data insights at each step of the way, I do not want to go through and delete these. For future logs, however, I will simply only look at the final four tables. I like to see the insights for the joint user and IP tables from Invalid. Beyond that, we will just look at the two final summary tables.

linux_2k.log For this file, we are going to look for four keywords: failure, unknown, opened, and connection. Opened is to find the strings for session opened, which implies that a user logs in. Unknown is for messages like "check pass; user unknown", where someone fails to log-in.

Check pass will simply be counted, as there is no IP or username to retrieve. Connection provides us with an IP and session provides us with a username. Unfortunately, there is no way to connect these two. Failure provides us with a user field, but this does not appear in every row.

```
head(linux_2k.log$message, n = 50)
```

```
##  [1] "authentication failure; logname= uid=0 euid=0 tty=NODEVssh ruser= rhost=218.188.2.4 "
##  [2] "check pass; user unknown"
##  [3] "authentication failure; logname= uid=0 euid=0 tty=NODEVssh ruser= rhost=218.188.2.4 "
##  [4] "authentication failure; logname= uid=0 euid=0 tty=NODEVssh ruser= rhost=220-135-151-1.hinet-ip
##  [5] "authentication failure; logname= uid=0 euid=0 tty=NODEVssh ruser= rhost=220-135-151-1.hinet-ip
##  [6] "authentication failure; logname= uid=0 euid=0 tty=NODEVssh ruser= rhost=220-135-151-1.hinet-ip
##  [7] "authentication failure; logname= uid=0 euid=0 tty=NODEVssh ruser= rhost=220-135-151-1.hinet-ip
##  [8] "authentication failure; logname= uid=0 euid=0 tty=NODEVssh ruser= rhost=220-135-151-1.hinet-ip
##  [9] "authentication failure; logname= uid=0 euid=0 tty=NODEVssh ruser= rhost=220-135-151-1.hinet-ip
## [10] "authentication failure; logname= uid=0 euid=0 tty=NODEVssh ruser= rhost=220-135-151-1.hinet-ip
## [11] "authentication failure; logname= uid=0 euid=0 tty=NODEVssh ruser= rhost=220-135-151-1.hinet-ip
## [12] "authentication failure; logname= uid=0 euid=0 tty=NODEVssh ruser= rhost=220-135-151-1.hinet-ip
## [13] "authentication failure; logname= uid=0 euid=0 tty=NODEVssh ruser= rhost=220-135-151-1.hinet-ip
## [14] "session opened for user cyrus by (uid=0)"
## [15] "session closed for user cyrus"
## [16] "ALERT exited abnormally with [1]"
## [17] "session opened for user news by (uid=0)"
## [18] "session closed for user news"
## [19] "check pass; user unknown"
## [20] "authentication failure; logname= uid=0 euid=0 tty=NODEVssh ruser= rhost=218.188.2.4 "
## [21] "check pass; user unknown"
## [22] "authentication failure; logname= uid=0 euid=0 tty=NODEVssh ruser= rhost=218.188.2.4 "
## [23] "check pass; user unknown"
## [24] "authentication failure; logname= uid=0 euid=0 tty=NODEVssh ruser= rhost=218.188.2.4 "
## [25] "check pass; user unknown"
## [26] "authentication failure; logname= uid=0 euid=0 tty=NODEVssh ruser= rhost=218.188.2.4 "
## [27] "check pass; user unknown"
## [28] "authentication failure; logname= uid=0 euid=0 tty=NODEVssh ruser= rhost=218.188.2.4 "
## [29] "check pass; user unknown"
## [30] "check pass; user unknown"
```

```
## [31] "check pass; user unknown"
## [32] "authentication failure; logname= uid=0 euid=0 tty=NODEVssh ruser= rhost=218.188.2.4 "
## [33] "check pass; user unknown"
## [34] "authentication failure; logname= uid=0 euid=0 tty=NODEVssh ruser= rhost=218.188.2.4 "
## [35] "authentication failure; logname= uid=0 euid=0 tty=NODEVssh ruser= rhost=218.188.2.4 "
## [36] "authentication failure; logname= uid=0 euid=0 tty=NODEVssh ruser= rhost=218.188.2.4 "
## [37] "check pass; user unknown"
## [38] "authentication failure; logname= uid=0 euid=0 tty=NODEVssh ruser= rhost=218.188.2.4 "
## [39] "check pass; user unknown"
## [40] "authentication failure; logname= uid=0 euid=0 tty=NODEVssh ruser= rhost=218.188.2.4 "
## [41] "check pass; user unknown"
## [42] "authentication failure; logname= uid=0 euid=0 tty=NODEVssh ruser= rhost=218.188.2.4 "
## [43] "check pass; user unknown"
## [44] "authentication failure; logname= uid=0 euid=0 tty=NODEVssh ruser= rhost=061092085098.ctinets.c
## [45] "check pass; user unknown"
## [46] "authentication failure; logname= uid=0 euid=0 tty=NODEVssh ruser= rhost=061092085098.ctinets.c
## [47] "check pass; user unknown"
## [48] "authentication failure; logname= uid=0 euid=0 tty=NODEVssh ruser= rhost=061092085098.ctinets.c
## [49] "check pass; user unknown"
## [50] "authentication failure; logname= uid=0 euid=0 tty=NODEVssh ruser= rhost=061092085098.ctinets.c
```

```
#Invalid
#failure
failure = grep("failure", linux_2k.log$message)
df_failure <- linux_2k.log$message[failure]
df_failure = strsplit(df_failure,' ')
#We need to get rid of the lines that do not have the user element.
df_failure_users <- lapply(df_failure, function(x) if(length(x) >= 10) x[10])
df_failure_users <- df_failure_users[sapply(df_failure_users, Negate(is.null))]
df_failure_users <- unlist(df_failure_users)
df_failure_users = strsplit(df_failure_users, '=')
df_failure_users <- lapply(df_failure_users, function(x) x[2])
df_failure_users <- unlist(df_failure_users)
sort(table(df_failure_users), decreasing = TRUE)
```

```
## df_failure_users
##  root guest  test
##   351    17     4
```

```
#351 instances of root, 17 of guest, and 4 of test

#unknown

unknown = grep("unknown", linux_2k.log$message)
df_unknown <- linux_2k.log$message[unknown]
length(df_unknown)
```

```
## [1] 118
```

```
#118 occurences of unknown users
```

```
#Valid

#opened

opened = grep("opened", linux_2k.log$message)
df_opened <- linux_2k.log$message[opened]
df_opened = strsplit(df_opened,' ')
df_opened_users <- lapply(df_opened, function(x) x[5])
df_opened_users <- unlist(df_opened_users)
sort(table(df_opened_users), decreasing = TRUE)
```

```
## df_opened_users
## cyrus  news  test  root
##    43    43    36     1
```

```
# 43 successes for cyrus and news; 36 successful sessions for test, and 1 successful session for root

#Connection

connection = grep("connection", linux_2k.log$message)
df_connection <- linux_2k.log$message[connection]
df_connection = strsplit(df_connection,' ')
df_connection_ip <- lapply(df_connection, function(x) x[3])
df_connection_ip <- unlist(df_connection_ip)
head(sort(table(df_connection_ip), decreasing = TRUE), n = 6)
```

```
## df_connection_ip
##   203.101.45.59    207.30.238.8  211.72.151.162   211.107.232.1  202.82.200.188
##              46              46              44              41              35
## 210.245.165.136
##              32
```

mac_2k.log As far as I can tell, this log file's messaging has no rhyme or reason to it. There are very little repeated patterns, and searches for user and IP keywords provided little information.

```
failed = grep("Failed ", ssh_2k.log$message)
error = grep("error", ssh_2k.log$message)
idx = grep("input_userauth_request", ssh_2k.log$message)
#I had troubles reading in Invalid, but idx catches every username that fails to connect; this was prov


df_idx <- as.data.frame(ssh_2k.log$message[idx])
df_error <- as.data.frame(ssh_2k.log$message[error])
df_failed <- as.data.frame(ssh_2k.log$message[failed])
names(df_idx) <- "X1"
names(df_error) <- "X1"
names(df_failed) <- "X1"

#error
```

```
pattern="([0-9]+\\.){3}[0-9]+"
ips =str_extract(df_error$X1,pattern)
ips = as.data.frame(ips)
head(sort(table(ips), decreasing = TRUE), n = 6)
```

ssh_2k.log

```
## ips
##   103.99.0.122 195.154.37.122
##             45              2
```

```
#fail
pattern="([0-9]+\\.){3}[0-9]+"
ips =str_extract(df_failed$X1,pattern)
ips = as.data.frame(ips)
head(sort(table(ips), decreasing = TRUE), n = 6)
```

```
## ips
##  183.62.140.253 187.141.143.180    103.99.0.122    112.95.230.3    5.188.10.180
##             286              80              45              26              20
##  185.190.58.151
##              18
```

```
#idx
username_idx <- sub(".*\\buser\\s+(\\w+| ).*", "\\1", df_idx$X1)
username_idx = as.data.frame(username_idx)
head(sort(table(username_idx), decreasing = TRUE), n = 6)
```

```
## username_idx
##   admin  oracle support    test    user       0
##      21       6       6       5       4       3
```

```
# This file has NO valid logins!
```

Sudo commands: auth.log

Auth2.log is the only table with sudo commands.

auth2.log Let's look at the executables/programs run by sudo.

```
sudo = auth2.log$message[auth2.log$app == "sudo"]
COMMAND = grep("COMMAND", sudo)
df_COMMAND = sudo[COMMAND]
split_COMMAND = strsplit(df_COMMAND, ' ')
values <- lapply(split_COMMAND, function(x) x[11])
values <- unlist(values)
commands = strsplit(values, '=')
commands <- lapply(commands, function(x) x[2])
commands <- unlist(commands)
length(unique(commands))
```

```
## [1] 30
```

```
sort(table(commands), decreasing = TRUE)
```

```
## commands
##                       /usr/bin/vim                 /usr/sbin/service
##                                 63                                18
##                          ./filebeat                  /usr/bin/apt-get
##                                 15                                12
##               /usr/bin/filebeat.sh                           /bin/su
##                                 12                                 9
##                     /sbin/auditctl                              root
##                                  6                                 5
##                ./create_n_users.sh                           /bin/rm
##                                  4                                 4
## /usr/share/filebeat/bin/filebeat                          /bin/chown
##                                  4                                 3
##                             /bin/cp                  /sbin/resolvconf
##                                  3                                 3
##                       /usr/bin/dpkg                     /usr/bin/tail
##                                  3                                 3
##                          /bin/chmod                        /bin/mkdir
##                                  2                                 2
##               /usr/bin/hostnamectl                        /usr/bin/vi
##                                  2                                 2
##              /usr/sbin/update-rc.d                     /bin/hostname
##                                  2                                 1
##                             /bin/ls                   /sbin/ausearch
##                                  1                                 1
##                       /usr/bin/apt                  /usr/bin/apt-key
##                                  1                                 1
##                       /usr/bin/curl                 /usr/bin/hexdump
##                                  1                                 1
##                       /usr/bin/tee                /usr/sbin/groupadd
##                                  1                                 1
```

Next, we will look at the users. The user pops up in two different places as either user or USER.

```
USER = grep("USER", sudo)
df_USER = sudo[USER]
user = grep("user", sudo)
df_user = sudo[user]

split_USER = strsplit(df_USER, ' ')
USER_values <- lapply(split_USER, function(x) x[[9]])
USER_values <- unlist(user_values)
unique(USER_values)
```

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

```
table(USER_values)["USER=root"]
```

```
## <NA>
##   NA
```

```
table(USER_values)["PWD=/usr/share/filebeat/scripts"]
```

```
## <NA>
##   NA
```

```
table(USER_values)["PWD=/home/ubuntu"]
```

```
## <NA>
##   NA
```

```
split_user = strsplit(df_user, ' ')
user_values <- lapply(split_user, function(x) x[[6]])
user_values <- unlist(user_values)
unique(user_values)
```

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

```
table(user_values)["root"]
```

```
## root
##  371
```

```
table(user_values)[";"]
```

```
## ;
## 5
```

```
split_user[split_user[[6]] == ";"]
```

```
## list()
```

```
matching_indices <- sapply(split_user, function(x) x[6] == ";")
matching_strings <- split_user[matching_indices]
head(matching_strings, n = 1)
```

```
## [[1]]
##  [1] ""                              ""
##  [3] "ubuntu"                        ":"
##  [5] "TTY=pts/0"                     ";"
##  [7] "PWD=/home/ubuntu/misc_scripts" ";"
##  [9] "USER=root"                     ";"
## [11] "COMMAND=./create_n_users.sh"
```

In the first batch of checking for usernames, I look for 'USER=" "' in the message fields corresponding to sudo actions. For these messages, every single user is root. I accidentally included some bad strings; however, they can be ignored due to their small size in comparison.

In the second batch, we get root for all users as well, with the exception of five coereced semicolons. When I looked at these five strings, we see that the User is still root; it is just located in a different place.

Lastly, let's look at the machine used. Sudo runs on the local host, so we just need the logging host itself. For file auth2.log, there is only one machine: ip-10-77-20-248. Return back

```
machine = auth2.log$`logging host`[auth2.log$app == "sudo"]
head(machine, n = 1)
```

```
## [1] "ip-10-77-20-248"
```

```
length(unique(machine))
```

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