Using python scripts for log analysis

Max Rochefort-Shugar – 02/03/18

Contents

[Introduction 3](#_Toc508288452)

[GUI 3](#_Toc508288453)

[Task 1 4](#_Toc508288454)

[Code 4](#_Toc508288455)

[Results 4](#_Toc508288456)

[Task 2 5](#_Toc508288457)

[Code 5](#_Toc508288458)

[Results 5](#_Toc508288459)

[Task 3 6](#_Toc508288460)

[Code 6](#_Toc508288461)

[Results 6](#_Toc508288462)

[Task 4 7](#_Toc508288463)

[Code 7](#_Toc508288464)

[Results 7](#_Toc508288465)

[Task 5 8](#_Toc508288466)

[Code 8](#_Toc508288467)

[Results 9](#_Toc508288468)

[Discussion 9](#_Toc508288469)

[References 10](#_Toc508288470)

# Introduction

The regular expression library will come in handy for this assignment since we are dealing with a lot of string manipulation. I will display snippets of code throughout this document relevant to each task and the results they produce. At the end of this document we will discuss our findings from the code and how I believe the attack took place. Inline comments have been removed from the code snippets displayed in this document due to the limited space within a text box. All code within the main program has been fully commented.

# GUI

I have created a GUI using the tkinter library which displays the results of tasks one to five. The code that initialises the GUI can be seen below.

rw **=** Tk**()**

rw**.**title**(**"Text File Parser"**)**

configfile **=** Text**(**wrap**=**WORD**,** width**=**60**,** height**=** 10**)**

configfile**.**pack**(**fill**=**X**,** padx**=**10**,** pady**=**10**)**

configfile**.**insert**(**INSERT**,** 'Welcome to the text file parser GUI.'**)**

btn1**=**ttk**.**Button**(**rw**,** text**=**"Display the number of failed login attempts for bin."**)**

btn1**.**pack**(**fill**=**X**,** padx**=**10**,** pady**=**10**)**

btn1**.**config**(**command**=**get\_bin\_attempts**)**

btn2**=**ttk**.**Button**(**rw**,** text**=**"Create and display blacklisttips.txt"**)**

btn2**.**pack**(**fill**=**X**,** padx**=**10**,** pady**=**10**)**

btn2**.**config**(**command**=**run\_blacklists**)**

btn3**=**ttk**.**Button**(**rw**,** text**=**"Calculate the number of attacks per hour."**)**

btn3**.**pack**(**fill**=**X**,** padx**=**10**,** pady**=**10**)**

btn3**.**config**(**command**=**get\_attack\_frequency**)**

btn4**=**ttk**.**Button**(**rw**,** text**=**"Calculate the number of attacks per hour per IP."**)**

btn4**.**pack**(**fill**=**X**,** padx**=**10**,** pady**=**10**)**

btn4**.**config**(**command**=**get\_attack\_ip**)**

btn5**=**ttk**.**Button**(**rw**,** text**=**"Detect Correllation"**)**

btn5**.**pack**(**fill**=**X**,** padx**=**10**,** pady**=**10**)**

btn5**.**config**(**command**=**compare\_files**)**

When the user presses a button, a function is called, and the results are displayed in a text box. The variable ‘configfile’ refers to the text box within the GUI which is used to display the results of each task.

# Task 1

**Find how many attempts were made with the bin account.**

The sentence that signifies a login attempt for the bin account is ‘Failed password for bin’.

## Code

file = open('auth.log', 'r').read()

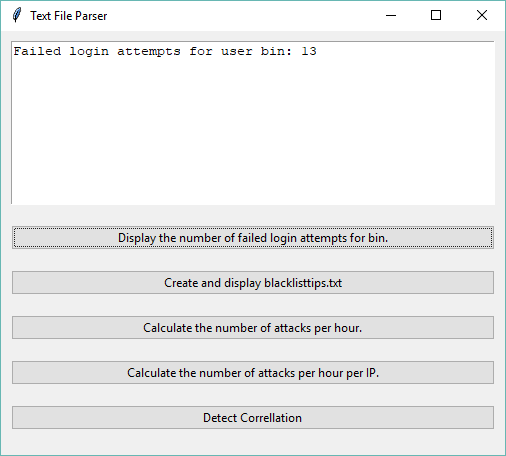
count = file.count("Failed password for bin")

configfile.delete('1.0', END)

configfile.insert(INSERT, "Failed login attempts for user bin: %d" % count)

The code above makes use of the file.count() function which counts the number of occurrences of the string ‘Failed password for bin’. configFile.delete() clears the the text box and configfile.insert() inserts data into the text box.

## Results



# Task 2

**If certain IP addresses have more than 30 failed attempts, create a blacklist file (blacklistips.txt) and save the IP addresses within it.**

## Code

Every IP address that appears in auth.log is stored as a key within the ‘occurrence’ dictionary. Every time the same IP address occurs, + 1 is added to the keys value. If the value of any key is greater than 30 then the key and value is stored in blacklisttips.txt.

def grab\_ip(file): #create a function called grap\_ip

with open("blacklisttips.txt", "w") as text\_file:

text\_file.write("")

occurrence = {}

with open (file) as file:#open file

for ip in file:

selected\_data = re.findall(r'Failed password for .\*from\* .\*(?:[\d]{1,3})\.(?:[\d]{1,3})\.(?:[\d]{1,3})\.(?:[\d]{1,3})\*', ip)#find attack string

for data in selected\_data:

ip\_s = re.findall(r'(?:[\d]{1,3})\.(?:[\d]{1,3})\.(?:[\d]{1,3})\.(?:[\d]{1,3})', data)#parse ip from attack

for ipaddr in ip\_s:

if ipaddr in occurrence:

occurrence[ipaddr] = occurrence[ipaddr] + 1

else:

occurrence[ipaddr] = 1

for key, value in occurrence.items():

if (value > 30):

with open("blacklisttips.txt", "a") as text\_file:

text\_file.write("%s had %d failed login attempts\n" % (key, value))

return None

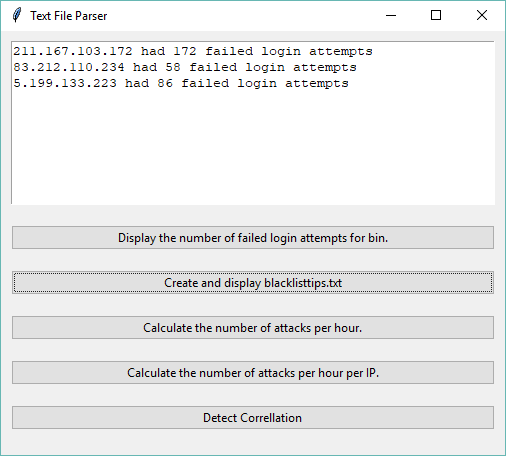
grab\_ip('auth.log')

configfile.delete('1.0', END)

with open('blacklisttips.txt', 'r') as f:

configfile.insert(INSERT, f.read())

## Results



# Task 3

**Identify how many attacks were logged per hour.**

## Code

attacks = []

counts = dict()

with open("auth.log") as f:

for line in f:

if "Failed password for" in line:

attacks.append(line[:9])

for i in attacks:

counts[i] = counts.get(i, 0) + 1

configfile.delete('1.0', END)

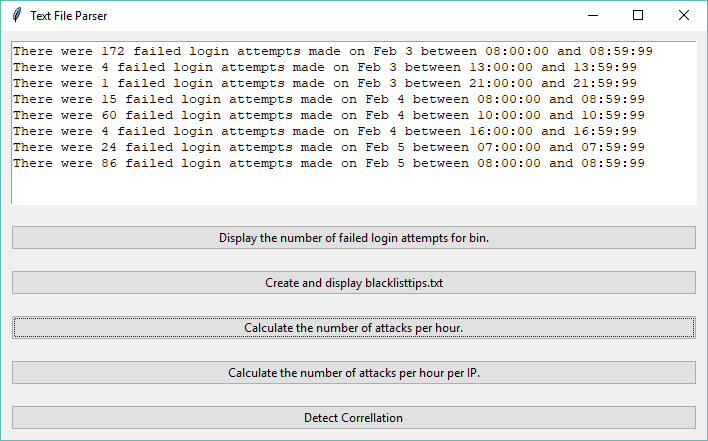
for key, value in counts.items():

month, day, hour = key[0:3], key[4:6], key[7:9]

configfile.insert(INSERT, "There were %d failed login attempts made on %s%s between %s:00:00 and %s:59:99\n" % (value, month, day, hour, hour))

If a line has the text ‘Failed password for’ then the first nine characters are stored in the ‘attacks’ array because the month, day and hour can be inferred from the first nine characters. The nine characters are stored as a key within the ‘counts’ dictionary. If the key does not have a value, then the default number 0 is used. If the key does have a value, then +1 is added to that value for every occurrence of the same 9 characters. This value represents the number of occurrences of failed login attempts per hour.

## Results



# Task 4

**Identify how many attacks were logged, per hour, per IP.**

Similarly, to the previous task, the hour, day and month can be inferred from the first nine characters of each line in auth.log. We can also retrieve the IP address from each line too.

## Code

attacks = (line for line in open('auth.log', 'r') if "Failed password for" in line)

configfile.delete('1.0', END)

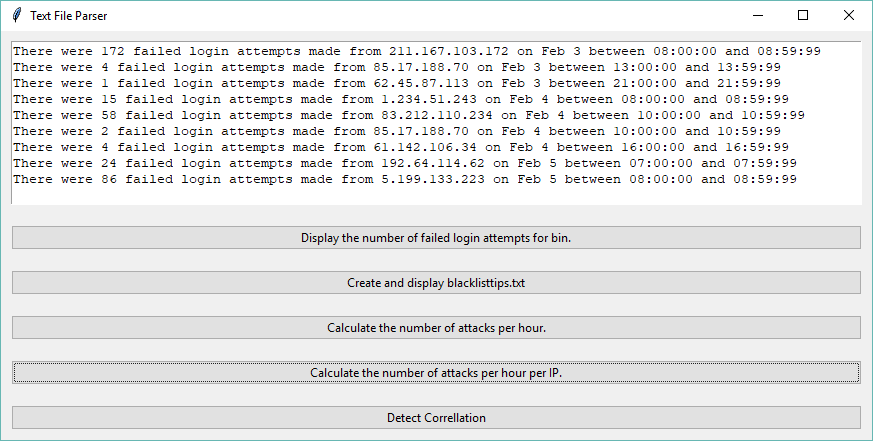
for key, group in groupby(attacks, key = lambda z: z[:9] + re.search('from(.+?) ', z).group()):

month, day, hour, ip = key[0:3], key[4:6], key[7:9], key[14:]

configfile.insert(INSERT, "There were %d failed login attempts made from %son %s%s between %s:00:00 and %s:59:99\n" % (len(list(group)), ip, month, day, hour, hour) )

Each line that contains the text ‘Failed password for’ is stored in the ‘’attacks’ variable. The first nine characters and IP address are grouped together. The number of attacks per IP can be inferred from the size of each group.

## Results



# Task 5

**Compare the results from the Webserver’s log and those from the SSH logs and create a new text file with a new entry describing the correlation of both events.**

My plan for this code was to compare the IP addresses in both files and see if there is a common occurrence.

## Code

configfile.delete('1.0', END)

webserver\_ip = {}

with open ('access.log') as file:#open file

for line in file:

temp = r'(?:[\d]{1,3})\.(?:[\d]{1,3})\.(?:[\d]{1,3})\.(?:[\d]{1,3})'

selected\_data = re.findall(temp, line)

for ip in selected\_data:

if ip in webserver\_ip:

webserver\_ip[ip] = webserver\_ip[ip] + 1

else:

webserver\_ip[ip] = 1

for key, value in webserver\_ip.items():

with open ('auth.log') as file:

for line in file:

if re.findall('(.+){0}(.+)'.format(key), line):

x = line.split('Accepted password for ')

y = x[1].split(' ')

username = y[0]

correlated\_line\_auth = line

with open ('access.log') as file:#open file

for line in file:

if re.findall(username, line):

#print(line)

x = line.split('username=jsmith&password=')

y = x[1].split(' ')

password = y[0]

correlated\_line\_access = line

with open("correlation.txt", "w") as text\_file:

text\_file.write("Username: %s Password: %s\n\n" % (username, password))

text\_file.write('Auth.log: %s\n' % correlated\_line\_auth)

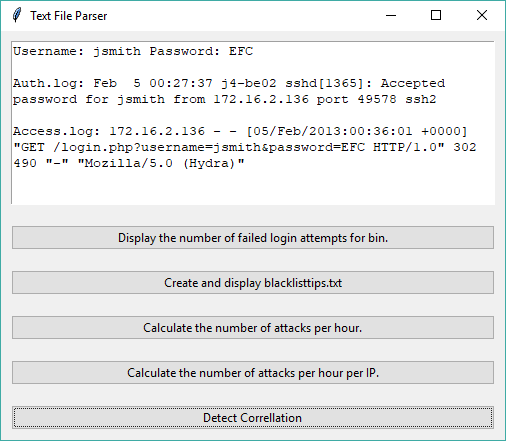
text\_file.write('Access.log: %s\n' % correlated\_line\_access)#write to the text file

with open('correlation.txt', 'r') as f:

configfile.insert(INSERT, f.read())

The IP addresses from access.log are stored in a dictionary. If the IP addresses from access.log appear in auth.log then there is a match. We can infer the username and password used to gain access to the system and from both files where there is a matching IP address. The lines from both files with the same IP address are stored a text file along with the username and password used to gain access to the system.

## Results



# Discussion

The username used to gain access to the system was ‘jsmith’ and the password was ‘EFC’. The username jsmith was used for every login attempt for access.log while the password was changed. A brute force attack is where automated software is used to generate many consecutive guesses of the value of the desired data (Techopedia, no date). The attacker’s algorithm appears to be generating passwords by incrementing letters of the alphabet continuously. The brute force algorithm might be using words from the dictionary to generate passwords while continually incrementing the letter furthest to the left. A dictionary attack is a method of breaking into password protected computer systems by systematically entering words from the dictionary (TechTarget, no date). I believe a brute force attack was used to gain access to the system. All entries within access.log occur within 1 second which supports the idea of automated software being used to generate many guesses within a small period that would have been unachievable by a single human being. The attacker used a Mozilla client to gain access to the system. Having done some research online, Hydra appears to be a penetration testing tool that is exclusively focused on dictionary-attacking web-based login forms (Hydra, no date). This reinforces what I have discussed above, being a dictionary attack being used. The attacker with IP 172.16.2.136 made 163 attempts to login to the system with the last attempt being a successful login.

I would suggest a few methods to protect against a brute force attack and other similar cybercriminal attacks. The organisations I.T. team can create a whitelist of IP addresses that have access to the system. IP addresses could be blacklisted once they’ve had several failed attempts or the username used could be locked so that the I.T. team will have to unlock it. Locking the username will mean that the attackers will have to go back to social engineering to obtain a new username or wait until the username is unlocked which will slow them down tremendously. In the case of the attack being discussed in this report, the username jsmith could have been locked a long time before 163 login attempts were made. Increased password complexity could be enforced by the I.T. Team so that users must create passwords with a combination of upper and lower-case characters, symbols and numbers. The length of the password makes it harder to crack too. Improved password requirements alone will make brute force attacks increasingly difficult to execute successfully.

# References

Techopedia https://www.techopedia.com/definition/18091/brute-force-attack (no date) (Accessed: 3rd March 2018)

TechTarget http://searchsecurity.techtarget.com/definition/dictionary-attack (no date) Accessed: 3rd March 2018)

Hydra https://github.com/opennota/hydra (Accessed: 5th March 2018)

Hydra https://tools.kali.org/password-attacks/hydra (Accessed: 5th March 2018)