Detection of attack using python project report in LaTeX

IIT Guwahati

→ **Shubham jindal (150101071)**
→ **Samrat yadav (150101056)**
→ **Sarthak tripathi (150101057)**

April 10, 2017

Index

Introduction

Computer Security is one of the most important aspects of Computer Science as in this modern age sensitive and important data is stored in computers. Its knowledge or manipulation could have catastrophic effects. To prevent attacks on computers to hack it, we must first detect attacks. System logs is an important way to detect attacks. A particular combination of numbers in the log file can imply there was an attack.

Example- multiple login/wrong password sequence of numbers can indicate the attempt to guess someone's password.

Thus finding such sequences can be useful in improving security features of computers and websites. One way to do so is find recurring sequences in known attacks. Thus these sequences can then help identify that attack. Thus documenting sequences with highest frequencies, it can help detect that attack, and this project uses this understanding and tries to detect attack.

Note:- Files are merged using command prompt using command merging files in folder :- **type *.txt >merged.txt**

Listing 1: Python code

```
1
2  import operator
3  import time
4  result=[]
5
6  def ngrams(input, n):
7    input = input.split(' ')
8    output = {}
9    for i in range(len(input)-n+1):
10     g = ' '.join(input[i:i+n])
11     output.setdefault(g, 0)
12     output[g] += 1
13   return output
14
15 def sort_ngrams(output):
16    input=sorted(output.items(),key=operator.itemgetter(1),reverse=True)
17   return input
18
19 def top30percent(x):
20   length=len(x)*3/10
21   count=1
22   for i in x:
23      if(count>length):
24        break;
25   result.append(i[0])
26   count=count+1
27
28 def main():
29
30   n=input("Enter n-gram value: ")
31
32     #This assumes that your program takes at least a tenth of second to run.
33  start_time = time.time() +0.1
34
35   Allfiles=[]
36   Adduser=("Adduser.txt","r")
37   Hydra_FTP=("Hydra_FTP.txt","r")
38   Hydra_SSH=("Hydra_SSH.txt","r")
39   Java_Meterpreter=("Java_Meterpreter.txt","r")
40   Meterpreter=("Meterpreter.txt","r")
41   Web_Shell=("Web_Shell.txt","r")
42   NormalData=("NormalData.txt","r")
43
44   Allfiles.append(Adduser)
45   Allfiles.append(Hydra_FTP)
46   Allfiles.append(Hydra_SSH)
47   Allfiles.append(Java_Meterpreter)
48   Allfiles.append(Meterpreter)
49   Allfiles.append(Web_Shell)
50   Allfiles.append(NormalData)
51
52   train_dataset=open("train.txt","w")
53
54   name=["Adduser","Hydra_FTP","Hydra_SSH","Java_Meterpreter","Meterpreter",
55         "Web_Shell","NormalData"]
56
57    join=[]
58
59
```

```
60
61    for j in Allfiles:                                        #Loop 1
62        output_temp=ngrams(open(j[0],j[1]).read(),n) # output_temp is dict
63        join.append(output_temp)    # join becomes list of dictionary
64        sorted_elements=sort_ngrams(output_temp)    #sort is list of tuples
65        top30percent(sorted_elements)
66
67    temp=[]
68    final=[]
69
70    for k in join: # seven times only                          #Loop 2
71        for i in result:
72            flag=k.get(i,0) #O(1)time by get() function
73            t=(i,flag)
74            temp.append(t)
75        final.append(temp)
76        temp=[]
77
78    index=0
79    string_header=str(n)+"-grams feature vectors:\n\n"
80    string_vector=""
81    for i in final: # seven times only                         #Loop 3
82        string1=""
83        string2=""
84        count=0
85        for j in i:
86            count=count+1
87            string2+=str(j[1])+" "
88            string1+="F"+str(count)+", "
89
90        string1+="-----> "
91        string2+=name[index]+"\n\n"
92        string_vector+=string1+string2
93        index+=1
94
95    final_vector=string_header+string_vector
96    train_dataset.write(final_vector)
97    print("--- %s seconds ---" % (time.time() - start_time))
98
99
100
101   main()
```

Loops

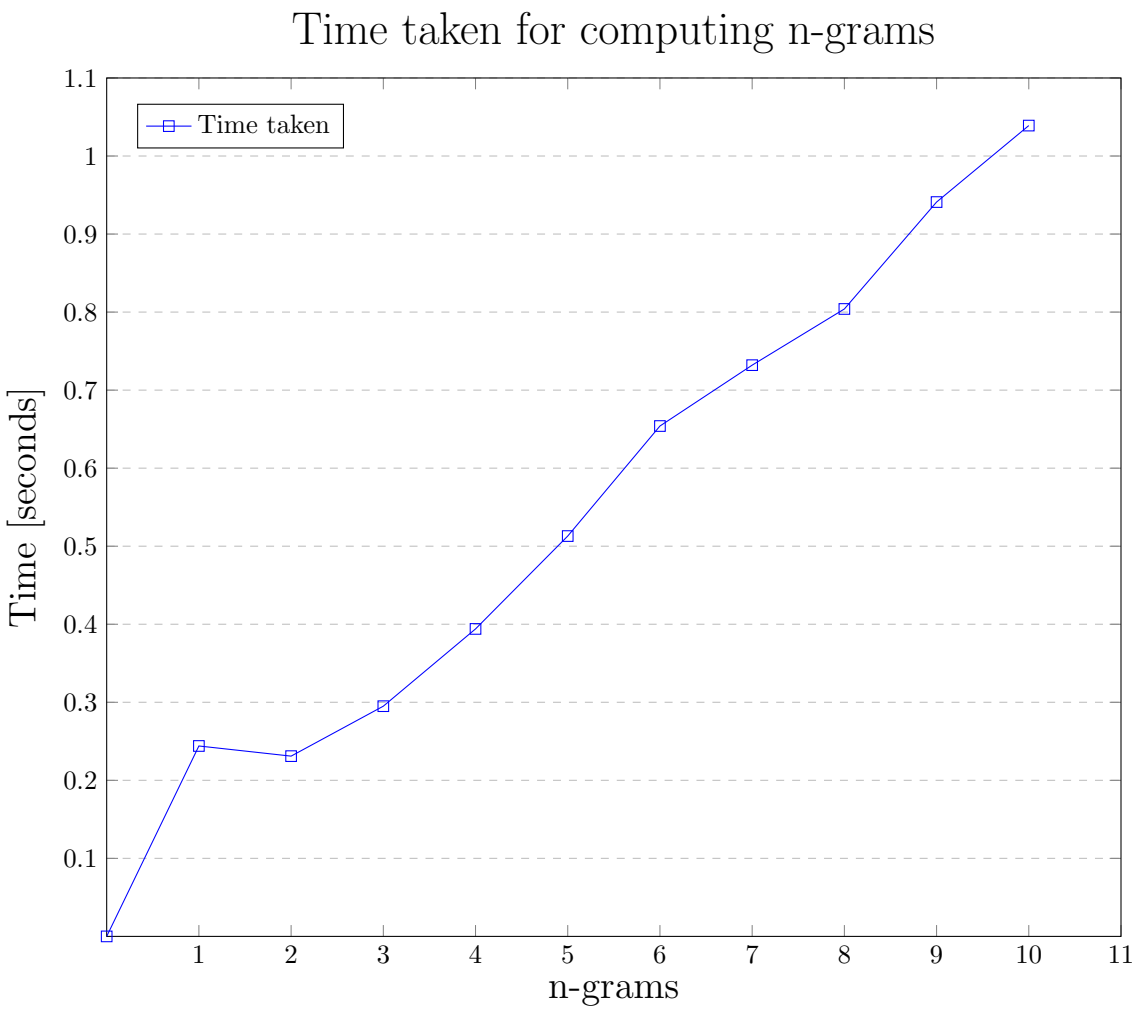
Loop	Discription	Complexity
Loop 1	all the log files are cocatenated into 'Allfiles' n-grams of Allfiles is sorted and n-grams of top 30% frequency is stored in join (list of Dictonary)	O(n logn)
Loop 2	top 30% of each type of log file is rechecked in all log file for frequency of their occuerence	O(n)
Loop 3	All of the frequncies are shown with their feature vector	O(n)

Functions

Function	Parameters	Output	Complexity
setdefault	key -This is the key to be searched. default-This is the Value to be returned in case key is not found	This method returns the key value available in the dictionary and Default value if given key is not available	O(n)
ngrams	input -values from log file n - n gram value	return n-grams in required format	O(n)
sortngram	output-unsorted n-gram	sorted n-grams in decreasing order of frequency	O(n log n)
top30percent	x -sorted n-grams in decreasing order of frequency	30% values from start	O(n)

Complexity of algorithm :- O(n log n)

1. ngrams(input,n)
    - Parameter- n -n gram value
    - input- values from log file
    - output- return n-grams in required format
    - complexity- O(n)
2. sort_ngrams(output)
    - Parameter- output -unsorted n gram
    - output- sorted n-grams in decreasing order of frequency
    - complexity- O(n log n)
3. top30percent(x)
    - Parameter- x -sorted n-gram in decreasing order of frequency
    - output- 30% values from start
    - complexity- O(n)


Time taken for computing n-grams
Time taken
Time [seconds]
n-grams
1.1
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
1
2
3
4
5
6
7
8
9
10
11