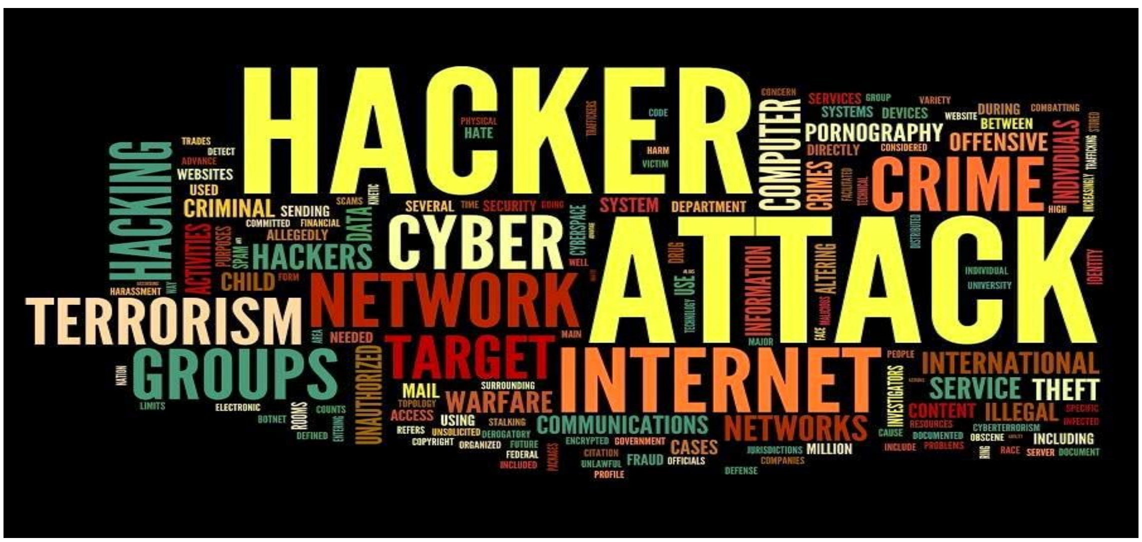
Detection of attack using python project report in LaTeX

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

```python
import operator
import time
result=[]
def ngrams(input, n):
    input = input.split(' ')
    output = {}
    for i in range(len(input)-n+1):
        g = ' '.join(input[i:i+n])
        output.setdefault(g, 0)
        output[g] += 1
    return output
# https://docs.python.org/3/library/operator.html
# Source for itemgetter() function
# Return a callable object that fetches item from its operand
# using the operands __getitem__() method.
# If multiple items are specified, returns a tuple of lookup values. For example:
# After f = itemgetter(2), the call f(r) returns r[2].
# After g = itemgetter(2, 5, 3), the call g(r) returns (r[2], r[5], r[3]).
# def itemgetter(*items):
#     if len(items) == 1:
#         item = items[0]
#         def g(obj):
#             return obj[item]
#     else:
#         def g(obj):
#             return tuple(obj[item] for item in items)
#     return g
def sort_ngrams(output):
    input=sorted(output.items(),key=operator.itemgetter(1),reverse=True)
    return input

def top30percent(x):
      length=len(x)*3/10
      count=1
      for i in x:
        if(count>length):
           break
        result.append(i[0])
        count+=1
def main():
    n=input("Enter n-gram value: ")

    #This assumes that your program takes at least a tenth of second to run.
    start_time = time.time() +0.1

    Allfiles=[]
    Adduser=("Adduser.txt","r")
    Hydra_FTP=("Hydra_FTP.txt","r")
    Hydra_SSH=("Hydra_SSH.txt","r")
    Java_Meterpreter=("Java_Meterpreter.txt","r")
    Meterpreter=("Meterpreter.txt","r")
    Web_Shell=("Web_Shell.txt","r")
    NormalData=("NormalData.txt","r")

    Allfiles.append(Adduser)
    Allfiles.append(Hydra_FTP)
    Allfiles.append(Hydra_SSH)
    Allfiles.append(Java_Meterpreter)
```

```
    Allfiles.append(Meterpreter)
    Allfiles.append(Web_Shell)
    Allfiles.append(NormalData)

    train_dataset=open("trained.txt","w")

    name=["Adduser","Hydra_FTP","Hydra_SSH","Java_Meterpreter","Meterpreter","
     Web_Shell","Normal"]

    names=["Adduser","Hydra_FTP","Hydra_SSH","Java_Meterpreter","Meterpreter",
     "Web_Shell"]

    Adduser=[]
    Hydra_FTP=[]
    Hydra_SSH=[]
    Java_Meterpreter=[]
    Meterpreter=[]
    Web_Shell=[]
    Normal=[]

    Adduser_vector=[]
    Hydra_FTP_vector=[]
    Hydra_SSH_vector=[]
    Java_Meterpreter_vector=[]
    Meterpreter_vector=[]
    Web_Shell_vector=[]
    Normal_vector=[]

    Adduser_vector_final=[]
    Hydra_FTP_vector_final=[]
    Hydra_SSH_vector_final=[]
    Java_Meterpreter_vector_final=[]
    Meterpreter_vector_final=[]
    Web_Shell_vector_final=[]
    Normal_vector_final=[]

    count=0
    for i in range(1,65):
        count+=1
        s="Adduser"+str(count)+".txt"
        Adduser.append(s)
        s=""
    count=0
    for i in range(1,115):
        count+=1
        s="Hydra_FTP"+str(count)+".txt"
        Hydra_FTP.append(s)
        s=""
    count=0
    for i in range(1,124):
        count+=1
        s="Hydra_SSH"+str(count)+".txt"
        Hydra_SSH.append(s)
        s=""
    count=0
    for i in range(1,88):
        count+=1
        s="Java_Meterpreter"+str(count)+".txt"
        Java_Meterpreter.append(s)
        s=""
```

```
    count=0
    for i in range(1,54):
        count+=1
        s="Meterpreter"+str(count)+".txt"
        Meterpreter.append(s)
        s=""
    count=0
    for i in range(1,84):
        count+=1
        s="Web_Shell"+str(count)+".txt"
        Web_Shell.append(s)
        s=""
    count=0
    for i in range(1,834):
        count+=1
        s="Normaldata"+str(count)+".txt"
        Normal.append(s)
        s=""

    for j in Allfiles:                                          #Loop 1
        output_temp=ngrams(open(j[0],j[1]).read(),n) # output_temp is dict
        sorted_elements=sort_ngrams(output_temp) #sort is list of tuples
        top30percent(sorted_elements)

    for i in Adduser:
        output_temp=ngrams(open(i,"r").read(),n)
        Adduser_vector.append(output_temp)
    for i in Hydra_FTP:
        output_temp=ngrams(open(i,"r").read(),n)
        Hydra_FTP_vector.append(output_temp)
    for i in Hydra_SSH:
        output_temp=ngrams(open(i,"r").read(),n)
        Hydra_SSH_vector.append(output_temp)
    for i in Java_Meterpreter:
        output_temp=ngrams(open(i,"r").read(),n)
        Java_Meterpreter_vector.append(output_temp)
    for i in Meterpreter:
        output_temp=ngrams(open(i,"r").read(),n)
        Meterpreter_vector.append(output_temp)
    for i in Web_Shell:
        output_temp=ngrams(open(i,"r").read(),n)
        Web_Shell_vector.append(output_temp)
    for i in Normal:
        output_temp=ngrams(open(i,"r").read(),n)
        Normal_vector.append(output_temp)

    temp=[]
    for k in Adduser_vector: # seven times only                 #Loop 2
        for i in result:
            flag=k.get(i,0) #O(1)time by get() function
            t=(i,flag)
            temp.append(t)
        Adduser_vector_final.append(temp)
        temp=[]

    for k in Hydra_FTP_vector: # seven times only
        for i in result:
            flag=k.get(i,0) #O(1)time by get() function
            t=(i,flag)
            temp.append(t)
```

```
        Hydra_FTP_vector_final.append(temp)
        temp=[]

    for k in Hydra_SSH_vector: # seven times only
        for i in result:
            flag=k.get(i,0) #O(1)time by get() function
            t=(i,flag)
            temp.append(t)
        Hydra_SSH_vector_final.append(temp)
        temp=[]

    for k in Java_Meterpreter_vector: # seven times only
        for i in result:
            flag=k.get(i,0) #O(1)time by get() function
            t=(i,flag)
            temp.append(t)
        Java_Meterpreter_vector_final.append(temp)
        temp=[]

    for k in Meterpreter_vector: # seven times only
        for i in result:
            flag=k.get(i,0) #O(1)time by get() function
            t=(i,flag)
            temp.append(t)
        Meterpreter_vector_final.append(temp)
        temp=[]

    for k in Web_Shell_vector:
        for i in result:
            flag=k.get(i,0) #O(1)time by get() function
            t=(i,flag)
            temp.append(t)
        Web_Shell_vector_final.append(temp)
        temp=[]

    for k in Normal_vector:
        for i in k:
            flag=k.get(i,0) #O(1)time by get() function
            t=(i,flag)
            temp.append(t)
        Normal_vector_final.append(temp)
        temp=[]

    feature_vector=[]
    feature_vector.append(Adduser_vector_final)
    feature_vector.append(Hydra_FTP_vector_final)
    feature_vector.append(Hydra_SSH_vector_final)
    feature_vector.append(Java_Meterpreter_vector_final)
    feature_vector.append(Meterpreter_vector_final)
    feature_vector.append(Web_Shell_vector_final)
    feature_vector.append(Normal_vector_final)

    index=0
    string_header=str(n)+"-grams feature vectors:\n\n"
    string_vector_final=""
    for k in feature_vector:                                    #Loop 3
        for i in k:
            string2=""
            count=0
            for j in i:
```

```
            string2+=str(j[1])+" "
        string2+=name[index]+"\n\n"
        string_vector_final+=string2
    index+=1

final_vector=string_header+string_vector_final
train_dataset.write(final_vector)
print("--- %s seconds ---" % (time.time() - start_time))
#print(final_string) # apply one tab here to get optiized format but that will disturb formatting

main()
```

Loops

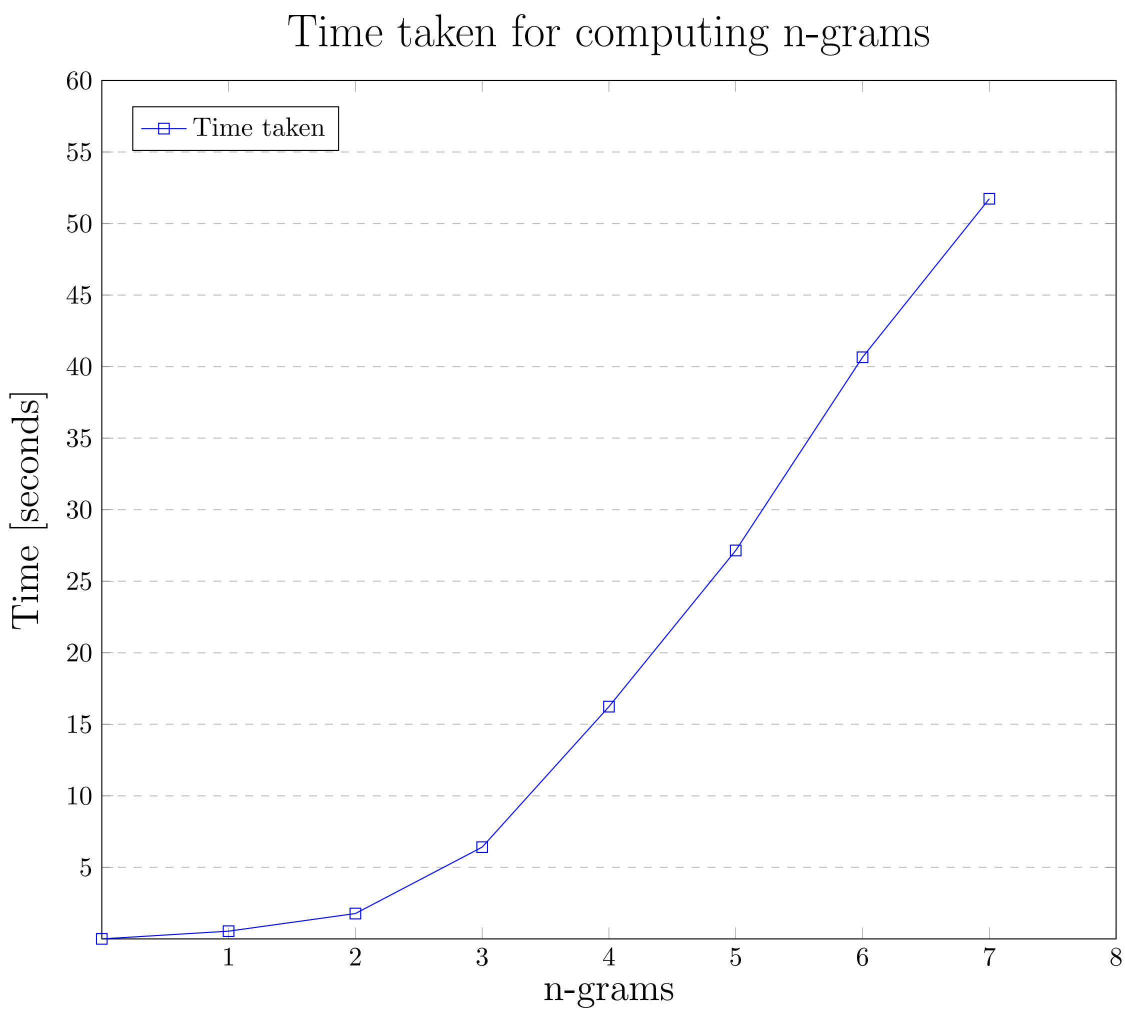
Loop	Discription	Complexity
Loop 1	all the log files are cocatenated into 'Allfiles' n-grams of Allfiles is sorted and n-grams of top 30% frequency is stored in join (list of Dictonary)	O(n logn)
Loop 2	top 30% of each type of log file is rechecked in all log file for frequency of their occuerence	O(n)
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)
top30percen	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
60
55
50
45
40
35
30
25
20
15
10
5
1
2
3
4
5
6
7
8