CS 244:-Software Programming
Project #1:-Computer Attacks

Due on Monday, April 10, 2017

Professor Santosh Biswas

Anurag Ramteke :- 150101010
Udayraj Deshmukh :- 150101021
Ayush Jain :- 150101012

Contents

Problem Statement 4

Libraries and Objects 4

Implementation 5

Libraries used 5

ngrams Array 5

Time Taken 5

Removing Files 5

Folders in Home Directory 6

Folders in Attack_Data_Master 6

Files in Attack_Data_Master 7

Training Data Loop 7

70% attackfolder selection 7

Gathering Training Data 8

Training Data Loop (Normal) 8

Gathering Normal Data 8

Concatenating Normal Data 9

Goto Main Directory 9

ngram Feature Loop 9

Creating Featurefiles 10

Training Files loop 10

ngrams(features) for Training Data 10

Counting Features 11

Sorting Features 11

Storing features in Files 11

30% Featurefile(Open) 12

30% Featurefile(Store) 12

Attack identifying function 12

ngram Unique Features loop **13**

Unique Featurefile(Open) **13**

Unique Feature(Arrray) **13**

String to Int(Features) **14**

Removing Duplicate features **14**

Unique Features(Store) **14**

Unique Features(Gathering) **15**

Folders in Home Directory_2 **15**

Output Directory(create) **15**

Folders in Attack_Data_Master_2 **16**

Output Attack_Data_Master folder(create) **16**

Test Folders **16**

Files in Attack_Data_Master_2 **16**

Output for Test Attack Files **17**

ngrams Loop for Test Attack_files **17**

Feature Loop for Test Attack_files **17**

Frequency of Features in Test Files(Output) **18**

Output(Store) **18**

Output Validation_Data_Master(Create) **18**

ngrams Loop for Validation_Data_Master Output **19**

Copy Particular Featurelist(Array) **19**

Dictionary of Features **19**

ngrams of Validation_Data_Master Files **20**

Frequency of Features in Validation_Data_Master Files **20**

Output Validation Files(Store) **20**

Output Validation Files(Store) **21**

Problem Statement

Given dataset was generated on Linux local server by Australian Defence Force Academy-Linux Dataset (ADFA-LD) running on Ubuntu 11.04, offering a variety of functions such as file sharing, database, remote access and web server.

Six types of attacks occur in ADFA-LD including two brute force password guessing attempts on the open ports enabled by FTP and SSH respectively, an unauthorised attempt to create a new user with root privileges through encoding a malicious payload into a normal executable, the uploads of Java and Linux executable Meterpreter payloads for the remote compromise of a target host, and the compromise and privilege escalation using C100 webshell. These types are termed as Hydra-FTP, Hydra-SSH, Adduser, Java-Meterpreter, Meterpreter and Webshell respectively.

So, to identify the type of attack from the given dataset, we implemented an algorithm to train the system from the Training_Data_Master and 70 % of the Attack_Data_Master and thus generating features from it. top 30% of these features(sorted in higher to lower frequency) is further used to find the frequency of these features in the test data (30%(remaining) of the Attack_Data_Master). This final data can be used to identify the type of attack.

So, the code implemented for it, is displayed and explained below.

Libraries and Objects

Libraries Used:- os,time,math,nltk

objects and functions used from these libraries:

os:

system(command) $-->$ works same as executing what is stored in command variable in terminal
getcwd() $-->$ returns the current working directory
walk(directory) $-->$ looping over walk with the directory as argument returns all the folders, subfolders and files in the directory
chdir(path) $-->$ changes the current working directory by taking the path to the another directory as argument

time: **time()** $-->$ returns value of time in seconds since 0 hours, 0 minutes, 0 seconds, January 1, 1970

math: **ceil()** $-->$ takes argument as a floating number and returns the nearest integer equal to or greater than it

nltk: **ngrams(text,so_grams)** $-->$ returns the ngrams of size given as so_grams as input for the data given in the text argument;

Implementation

Libraries used

Listing 1: Shows all libraries and some objects of them

```
import os
import time
import math
import nltk
from time import time
from nltk import ngrams
```

It imports all the libraries and some necessary objects as well as functions from it.

ngrams Array

Listing 2: List of ngrams to be computed

```
all_net_grams=[3,5,7]
```

This variable stores the size of grams for which the training and testing is to be done, it can be changed to any number large or small greater than zero and length of the variable (*list*) can be set as long as wanted. However, the more the arguments or bigger the number, more will be the time taken for the execution of the program. e.g. 3,5,7 are the size of ngrams with the length of the variable becoming equal to 3.

Time Taken

Listing 3: Time taken by code

```
line 0              timestamp=time();
 .
 .
 .
 .                  /*code*/
 .
 .
line t              print("time taken = ",time()-timestamp)
```

This starts counting time at line 0 and at *linet* prints the time taken to execute the code between those lines(*line*0 and *linet*).

Removing Files

Listing 4: Removing unnecessary Files

```
os.system("rm hydra_ftp_training.txt & . . . . . & rm webshell_training.txt")

os.system("rm Training_Data_Master.txt & . . . . . & rm hydra_ftp_test.txt ");
os.system("rm hydra_ssh_test.txt & . . . . .& rm webshell_test.txt")
os.system("rm -rf output")
os.system("rm featurefile.txt");
os.system("rm 3* & . . . . .& rm 9*")
```

This deletes all the unnecessary files and folders and the files and folders generated during the running of the last instance of the program.

Folders in Home Directory

Listing 5: Getting folders in the main directory

```
cwd0=os.getcwd();
for sub0,folders0,files0 in os.walk(cwd0):
     if len(folders0)>0 :
          break;
folders0.sort();
```

*cwd*0 stores the current working directory which is ADFA-LD directory where Attack_Data_Master and Training_Data_Master are present, we will call this directory as main directory of ADFA-LD directory. In the next set of lines, we recursed over all the files and folders in the main directory and then the array($folders0$) stores folders(at that time i.e. when $folders0$ stores folders $len(folders) > 0$) then it will break. This part is necessary since, we are recursing over all the files and folders and at most time, we will be recursing over files inside the main directory and at that time folders0 will be empty. After getting all the folders, we sorted the folders in alphabetical order. (Note: $folders0$, *sub*0, *cwd*0 the numbers succedding the variables here are 0 which represents the depth of those directory,files or folders with respect to the main directory.)

Folders in Attack_Data_Master

Listing 6: Getting folders' list in Attack_Data_Master

```
path=cwd0+"/"+folders0[0];
os.chdir(path);
cwd1=os.getcwd();
for sub1,folders1,files1 in os.walk(cwd1):
     if len(folders1)>0:
          break;
folders1.sort();
```

Here, we changed the directory with os.chdir with the input argument as path. We set path by using the current directory which is the main directory here and then appending the folders[0] to it, since it is sorted in alphabetical order it is Attack_Data_Master. so cwd1 now stores the path to the Attack_Data_Master. We then stored all the names of folders inside Attack_Data_Master in the folders1 in alphabetical order. (Notice:here, the depth of folders and directories with respect to the main directory is 1 so the name given to them is folders1, *cwd*1...).

Files in Attack_Data_Master

Listing 7: Getting files' list in Attack_Data_Master

```
path=cwd1+"/"+folders1[i];
os.chdir(path);
cwd2=os.getcwd();
for sub2,folders2,files2 in os.walk(cwd2):
          pass;
```

Path consists the i^{th} folder in $cwd1$ i.e. Attack_Data_Master in this case. Then change the current working directory to that i^{th} folder.$cwd2$ contains the path to this directory. All the files in that folder will be stored in files2.

Training Data Loop

Listing 8: Training Data Loop

```
for i in range(0,60):
     .
     .
     Listing 9
     Listing 10
     .
     .
```

Listing 9 and listing 10 will be looped using this command and i will be determined by the loop number (i will start from 0).

70% attackfolder selection

Listing 9: 70% attackfolders from Attack_Data_Master

```
if i<10:
     if i%10==1 or i%10==8 or i%10==9:
          location=cwd0+"/adduser_test.txt"
     else :
          location=cwd0+"/adduser_training.txt";
elif i<20:
     if i%10==1 or i%10==8 or i%10==9:
          location=cwd0+"/hydra_ftp_test.txt"
     else :
          location=cwd0+"/hydra_ftp_training.txt";
elif i<30:
     if i%10==1 or i%10==8 or i%10==9:
          location=cwd0+"/hydra_ssh_test.txt"
     else :
          location=cwd0+"/hydra_ssh_training.txt";
elif i<40:
     if i%10==1 or i%10==8 or i%10==9:
          location=cwd0+"/java_meterpreter_test.txt"
     else :
```

70% attackfolder selection continued on next page...

```
        location=cwd0+"/java_meterpreter_training.txt";
elif i<50:
    if i%10==1 or i%10==8 or i%10==9:
        location=cwd0+"/meterpreter_test.txt"
    else :
        location=cwd0+"/meterpreter_training.txt";
else:
    if i%10==1 or i%10==8 or i%10==9:
        location=cwd0+"/webshell_test.txt"
    else :
        location=cwd0+"/webshell_training.txt";
```

Depending on the sorted list of files, location of files can be determined and thus from that according to that position (i), the name of the file will be determined which has to be used to add data from the given file.

Gathering Training Data

Listing 10: Reading data from attack files to generate training data

```
for data in range(0,len(files2)):
    fr=open(files2[data],"r")
    text=fr.read();
    fr.close();
    fw=open(location,"a");
    fw.write(text);
    fw.write("\n\n")
    fw.close();
```

Now, we will loop over all the files whose name is stored in $files2$ in the i^{th} directory mentioned above. In every loop, one file not already read will be opened and its data will be stored in text and will be closed. Then another file whose name is stored in location will be opened and text will be appended to it and then will be closed.

Training Data Loop (Normal)

Listing 11: Training data loop (normal)

```
for i in range(1,3):
.
.
Listing 12
Listing 13
.
.
```

Listing 12 and Listing 13 will be put under this loop.

Gathering Normal Data

Listing 12: Gathering normal data

```
path=cwd0+"/"+folders0[i];
os.chdir(path)
cwd1=os.getcwd();
for sub1,folders1,files1 in os.walk(cwd1):
    pass;
files1.sort();
```

path contains the path to the folder, here it is Training_Data_Master and Validation_Data_Master, it will change the current directory to each one iteratively and then get the path stored in *cwd*1. In each folder files in those folder will be stored in files1 in sorted order.

Concatenating Normal Data

Listing 13: Concatenating normal data

```
for data in range(0,len(files1)):
    fr=open(files1[data],"r");
    text=fr.read();
    fr.close;
    if i==1:
        location=cwd0+"/Training_Data_Master.txt";
    elif i==2:
        location=cwd0+"/Validation_Data_Master.txt";
    fw=open(location,"a");
    fw.write(text);
    fw.write("\n\n");
    fw.close();
```

Iteratively it goes to each file and then stores data in text from that file and set the location according to the i^{th}(loop number) as mentioned in the next section. opening the file in main directory according to the value of i, the data will be stored in that file.

Goto Main Directory

Listing 14: Goto Main Directory

```
os.chdir(cwd0);
training=["adduser_training.txt", . . . . . . ,"Training_Data_Master.txt"]
```

Current directory is changed to main directory and training list is initiated with the attack files and normal files.

ngram Feature Loop

Listing 15: ngram feature loop

```
for n in all_net_grams:
    .
```

ngram Feature Loop continued on next page. . .

```
    .
    listing 16
    listing 17
    .
    .
featurefile.close();
```

Iterate over all_net_grams (mentioned in Listing 2) and featurefiles created in the loop will be closed after the end of the loop.

Creating Featurefiles

Listing 16: Creating Featurefiles

```
fname=str(n)+"grams_featurefile.txt"
featurefile=open(fname,"a")
```

Sets the name of $featurefile$ and open it. $featurefile$ consists of feature of ngrams as mentioned in $fname$.

Training Files loop

Listing 17: Training files loop

```
for t in training:
    .
    .
    Listing 18
    Listing 19
    Listing 20
    Listing 21
    Listing 22
    Listing 23
    .
    .
```

Iterates over concatenated file over all the attack files($6 attacks + 1 normal = total 7 files$).

ngrams(features) for Training Data

Listing 18: ngrams(features) for Training Data

```
length_of_trianing=len(training)
filename=str(n)+"gramsfor"+t
fw=open(filename,"a")
fr=open(t,"r")
text=fr.read();
fr.close();
all_grams=ngrams(text.split(),n)
```

Calculates the $length\_of\_training$, opens a file with name stored in the variable $filename$ and (attack file or normal file)will be opened and data from it will be stored in $text$ and then all the n grams depending on value of n will be stored in all_grams.

Counting Features

Listing 19: Counting features

```
for grams in all_grams:
    try:
        if point_int[grams]>=0:
            passing=point_int[grams];
    except:
        passing=-1;
    if passing>=0:
        ind=passing;
        count[ind]=count[ind]+1;
    else:
        data.append(grams)
        count.append(1)
        point_int[grams]=index_count;
        index_count=index_count+1;
```

Iterating over all the grams in *all_grams* generated in Listing 18, check if *point_int* dictionary has grams, if yes then *passing* is set to the dictionary value of that *grams*(which is the index of the feature in the *data*) else *passing* is set to -1. Then if the if condition is passed for $passing >= 0$ then increase the count in the *count* array where the i^{th} count corresponds to the i^{th} feature stored in *data*. If the condition fails then that *grams* is appended to the *data*, *count* to it is set to 1, new data to dictionary *point_int* is added with the value of its index in data and the *index_count* is increased by one which keeps the check on the index of the new value which will be stored in *data*.

Sorting Features

Listing 20: Counting features according to the frequency

```
p_data=[]
for i in range(len(data)):
    temp=[count[i]]+[data[i]]
    p_data.append(temp)
p_data.sort(reverse=True)
```

A list is initiaited *p_data* which is multidimensional whose length will be equal of no of features in *data* and each block in those feature consists of count of the feature and the feature which will be then sorted in reverse order(decreasing order).

Storing features in Files

Listing 21: Storing features in files

```
for j in range(len(p_data)):
    fw.write(str(p_data[j][0]));
    fw.write(" ")
    for i in range(n):
        fw.write(p_data[j][1][i]);
        fw.write(" ")
    fw.write("\n")
fw.close();
```

Then all the data from *p_data* is stored in file whose name was initiated in Listing 18.

30% Featurefile(Open)

Listing 22: Opening files for 30 percent of features

```
filename2="30percentwithout_count_of"+filename
filename="30percentof"+filename
fw=open(filename,"a")
fw2=open(filename2,"a")
```

These files are opened where 30% of top data from p_data is stored in $filename$ and $filename2$ but only difference between them is that $filename2$ does not tell the count of the features.

30% Featurefile(Store)

Listing 23: Opening files for 30 percent of features

```
for j in range(math.ceil((len(p_data))*0.3)):
    fw.write(str(p_data[j][0]));
    fw.write(" ")
    for i in range(n):
        fw.write(p_data[j][1][i]);
        fw2.write(p_data[j][1][i]);
        featurefile.write(p_data[j][1][i]);
        featurefile.write(" ")
        fw2.write(" ")
        fw.write(" ")
    fw.write("\n")
    featurefile.write("\n")
    fw2.write("\n")
fw.close();
```

Data is written from the files opened in Listing 16 and Listing 22.

Attack identifying function

Listing 24: Function to identify attack from folder name

```
def attack_type(foldfile):
    if "Adduser" in foldfile:
        return "Adduser";
    elif "Hydra" in foldfile:
        if "FTP" in foldfile:
            return "Hydra_FTP";
        elif "SSH" in foldfile:
            return "Hydra_SSH"
    elif "Meterpreter" in foldfile:
        if "Java" in foldfile:
            return "Java_Meterpreter";
        else :
            return "Meterpreter";
    elif "WS" in foldfile or "Web" in foldfile:
        return "Web_Shell"
    else:
        return "normal"
```

According to the name of the folder it returns the name of the attack.

ngram Unique Features loop

Listing 25: ngrams loop for finding unique features

```
for n in all_net_grams:
    .
    .
    Listing 26
    Listing 27
    .
    .
```

Iterates over *all_net_grams* as initiated in Listing 2.

Unique Featurefile(Open)

Listing 26: Unique featurefile(open)

```
fname=str(n)+"grams_featurefile.txt"
filename=str(n)+"grams_features.txt";
fr=open(fname,"r")
fw=open(filename,"a");
text=fr.read();
text=text.split("\n")
```

Stores the data from file with name stored in *fname* and splits the data to separate the data in different lines and store them again in *text* as array. opens a file with name stored in *filename*. This file will store all the unique n gram features.

Unique Feature(Arrray)

Listing 27: Storing all unique features in a loop

```
for i in range (len(text)):
    .
    .
    Listing 28
    Listing 29
    Listing 30
    .
    .
features.append(temp);
fr.close();
fw.close();
```

This will loop over the size of the *text* as generated in Listing 26 and *i* will be count on the loop number and files opened and created will be closed at last. *temp*(is mentioned in Listing 30) array will be appended to the *features* array.

String to Int(Features)

Listing 28: Converting data type of feature from string to int

```
text[i]=text[i].split();
for j in range(n):
    if i==len(text)-1:
        continue;
    text[i][j]=int(text[i][j]);
```

The *text* array created in Listing 26 will be further divided into subarrays based on the empty space between them and the individual strings will be converted into integer after that(features are divided into individual value and then converted to int).

Removing Duplicate features

Listing 29: Checking the repetition of features

```
ind=str(text[i])
try:
    if mydictionary[ind]==1:
        flag=1
except:
    flag=0;
if flag>0:
    continue;
```

The features are stored in *ind* and then check if the *mydictionary* has if, if yes then *flag* is turned to 1 else 0. so if the *mydictionary* has it(that particular feature) then next loop will start ending the currrent loop(we are removing duplicates here)

Unique Features(Store)

Listing 30: Storing the unique features in a file

```
mydictionary[ind]=1
temp.append(text[i])
for k in range(n):
    if i==len(text)-1:
        continue;
    fw.write(str(text[i][k]))
    fw.write(" ")
fw.write("\n")
```

If from the Listing 29 if the *flag* is greater than 1 then *mydictionary* with the feature in it has its value equal to 1, the feature will then be appended to the *temp*. Depending on the n grams(n was initiated in Listing 25) the data will be written into the file which was opened into Listing 26.

Unique Features(Gathering)

Listing 31: Gathering all the unique features

```
for i in all_net_grams:
    filename=str(i)+"grams_features.txt";
    fr=open(filename,"r");
    text=fr.read();
    text=text.split("\n")
    fr.close()
    featurelist.append(text)
cwd0=os.getcwd();
```

Iterating over *all_net_grams* (initiated in Listing 2), the unique n grams features(here *i*) will be stored in *text* and then split into on the basis of the line and that will be appended in *featurelist*. Then stores the current working directory is *cwd*0.

Folders in Home Directory_2

Listing 32: Getting folders in the main Directory

```
for sub0,fold0,file0 in os.walk(cwd0):
    if len(fold0)>0:
        break;
```

the data about the contents in the current folder is stored in *fold*0 which is the main directory.

Output Directory(create)

Listing 33: Creating output directory

```
os.system("mkdir output")
fold0.sort();
cwd1=cwd0+"/"+fold0[0];
os.chdir(cwd1)
```

Created output directory in the main directory, sorted the folders stored in *fold*0(it does not contain output folder). changed the directory to the first vlaue stored in *fold*0 which is Attack_Data_Master(in this case)

Folders in Attack_Data_Master_2

Listing 34: Getting folders list in Attack_Data_Master

```
for sub1,fold1,file1 in os.walk(cwd1):
    if len(fold1)>0:
    break;
```

Getting the list of folders from the current workign directory which is Attack_Data_Master(in this case)

Output Attack_Data_Master folder(create)

Listing 35: creating Attack_Data_Master folder in output directory

```
fold1.sort()
out_dir=cwd0+"/output"
os.chdir(out_dir)
os.system("mkdir Attack_Data_Master")
```

Sorting the list of folders whose names are stored in $fold1$. Changed the directory to the output folder in the main directory and created Attack_Data_Master folder in that directory.

Test Folders

Listing 36: Selecting folders for testing

```
for folder in fold1:
    filetype=attack_type(folder);
    if '8' in folder  or '9' in folder or '10' in folder:
        .
        .
        Listing 37
        Listing 38
        .
        .
```

$fold1$ (same from Listing 35) is iterated and in every iteration the value in the $fold1$ is stored in folder variable. According to the name of the folder filetype stores the attack type(Listing 24) and checking whether the folder is from last 3 folders in the list.

Files in Attack_Data_Master_2

Listing 37: Getting files list in Attack Data Master

```
out_dir=cwd0+"/output/Attack_Data_Master"
cwd2=cwd1+"/"+folder
os.chdir(cwd2);
for sub2,fold2,file2 in os.walk(cwd2):
    if len(file2)>0:
        break;
    pass;
sizeoffiles=len(file2)
```

Files in Attack_Data_Master_2 continued on next page...

out_dir variable stores the path to Attack_Data_Master folder in output directory. Then go to the folder stored in *folder* variable from Listing 36. *files*2 stores the name of all the files in that directory and *sizeoffiles* contains the number of files in that directory.

Output for Test Attack Files

Listing 38: Frequency of features from Attack_Data_Master test files

```
for files in file2:
    .
    .
    Listing 39
    .
    .
```

Iterates over all the files whose name is stored in *file*2 from Listing 37.

ngrams Loop for Test Attack_files

Listing 39: ngrams loop for frequency of features of test attack files

```
fr=open(files,"r")
text=fr.read()
fr.close();
for i in range(len(all_net_grams)):
    .
    .
    Listing 40
    Listing 41
    Listing 42
    .
    .
```

Open the files over which Listing 37 is iterating. Store the value of that file in *text* and then after closing. Iterating over *all_net_grams* starts(Listing 2).

Feature Loop for Test Attack_files

Listing 40: Feature loop for frequency of features of test attack files

```
featureindex=i;
thisfeatures=featurelist[featureindex];
count_features=0
for every_feature in thisfeatures:
    mydictionary[every_feature]=count_features;
    count_features=count_features+1
```

featurelist contains all the features of all grams and this features contains the particular features according to the loop i.e. the value of *i* as *i* increase the next value in *all_net_grams* is the ngrams features which will be stored in *thisfeatures*. Then a dictionary is implemented by name *mydictionary* which stores features in it and the value of those features is number of features appeared till that point.

Frequency of Features in Test Files(Output)

Listing 41: Frequency of each feature in the test files

```
single_grams=ngrams(text.split(),all_net_grams[i])
for grams in single_grams:
    string=grams[0]+" ";
    for j in range(1,all_net_grams[i]):
        string=string+grams[j]+" ";

    try:
        if mydictionary[string]>=0:
            flag=1;
    except:
            flag=0;
    if flag==1:
        ind=mydictionary[string];
        count[ind]=count[ind]+1;
```

Created ngrams and stored in *single_grams* for the data that was gathered from the last 3 folders(Listing 39). Iterating over all those grams, *string* consist the feature as a string in a particular format (e.g.:- "2 3 4 "). then if *mydictoinary* has that feature then $flag = 1$ else $flag = 0$. if $flag$ is 1 then after getting value of the string stored in *mydictionary* in *ind* we increase the count for that index with value of *ind* in *count* list(value of feature in *mydictionary* is the feature number)

Output(Store)

Listing 42: Storing output in files

```
filename=str(all_net_grams[i])+"gramsfrequency_"
out_dir=cwd0+"/output/Attack_Data_Master/";
os.chdir(out_dir)
fw=open(filename,"a")
for frequency in range(len(count)):
    fw.write(str(count[frequency]))
    fw.write(",")
fw.write(filetype)
fw.write("\n\n")
os.chdir(cwd2)
```

Changes directory to Attack_Data_Master and open the file with the file name as stored in $filename$ and write the frequency of those features in that file(features are generated from last 3 folders of Attack_Data_Master which is the test data for the attacks). Store that file in Attack_Data_Master which is present in output directory.

Output Validation_Data_Master(Create)

Listing 43: creating Validation_Data_Master folder in output directory

```
cwd1=cwd0+"/"+fold0[2];
```

```
out_dir=cwd0+"/output"
os.chdir(out_dir)
os.system("mkdir Validation_Data_Master")
os.chdir(cwd1)
for sub1,fold1,files1 in os.walk(cwd1):
    pass;
filetype=attack_type(fold0[2]);
```

Change the current working directory to output and make Validation_Data_Master folder in it. Get the name of all the files from the Validation_Data_Master in the main directory. *filetype* consists the name of attack if it is else normal according to the folder name.

ngrams Loop for Validation_Data_Master Output

Listing 44: ngrams loop for output for Validation_Data_Master

```
for files in files1:
    fr=open(files,"r")
    text=fr.read();
    fr.close();
    for i in range(len(all_net_grams)):
        .
        .
        Listing 45
        Listing 46
        Listing 47
        Listing 48
        Listing 49
        .
        .
```

Iteratively open the file and stores its data in *text*. Iterates over *all_net_grams*

Copy Particular Featurelist(Array)

Listing 45: Copy particular featurelist(array)

```
featureindex=i;
mydictionary={};
thisfeatures=featurelist[featureindex];
count_features=0
```

thisfeatures contains the feature list for particular ngrams. *featurelist* stores all the features.

Dictionary of Features

Listing 46: Dictionary of features

```
for every_feature in thisfeatures:
    mydictionary[every_feature]=count_features;
    count_features=count_features+1
```

Dictionary of Features continued on next page. . .

Makes *mydictionary* dictionary and stores all the features with its value equal to the index of feature number. $count\_features$ keeps the track of the feature index of size of *mydictionary*.

ngrams of Validation_Data_Master Files

Listing 47: ngrams of Validation_Data_Masater files

```
count=[0 for x in range(len(featurelist[featureindex])) ]
single_grams=ngrams(text.split(),all_net_grams[i]);
```

single_grams contains all the ngrams stored in *text*(Listing 44) according to the value in *all_net_grams* i^{th} index

Frequency of Features in Validation_Data_Master Files

Listing 48: Frequency of features in Validation_Data_Masater files

```
for grams in single_grams:
    string=grams[0]+" ";
    for j in range(1,all_net_grams[i]):
         string=string+grams[j]+" ";
    try:
        if mydictionary[string]>=0:
            flag=1;
    except:
        flag=0;
    if flag==1:
        ind=mydictionary[string];
        count[ind]=count[ind]+1;
```

Iterating over all those grams(Listing 44), *string* consist the feature as a string in a particular format (e.g.:- "2 3 4 "). then if *mydictoinary* has that feature then $flag = 1$ else $flag = 0$. if $flag$ is 1 then after getting value of the string stored in *mydictionary* in *ind* we increase the count for that index with value of *ind* in *count* list(value of feature in *mydictionary* is the feature number)

Output Validation Files(Store)

Listing 49: Output Validation_Data_Masater files(Store)

```
filename=str(all_net_grams[i])+"gramsfrequency"
out_dir=cwd0+"/output/Validation_Data_Master"
os.chdir(out_dir)
fw=open(filename,"a")
for frequency in range(len(count)):
    fw.write(str(count[frequency]))
    fw.write(",")
fw.write(filetype)
fw.write("\n\n")
os.chdir(cwd1)
```

Output Validation Files(Store) continued on next page...

Changes directory to Validation_Data_master and open the file with the file name as stored in *filename* and write the frequency of those features in that file(features are generated from Validation_Data_Master which is the test data for the attacks). Store that file in Validation_Data_masater in output.

Output Validation Files(Store)

Listing 50: Output Training_Data_Master files(Output)

```
for files in files1:
     fr=open(files,"r")
     text=fr.read();
     for i in range(len(all_net_grams)):
          mydictionary={};
          thisfeatures=featurelist[i];
          count_features=0
          mydictionary=dictionaryarray[i]
          count=[0 for x in range(len(featurelist[i])) ]
          single_grams=ngrams(text.split(),all_net_grams[i]);
          for grams in single_grams:
               string=grams[0]+" ";
               for j in range(1,all_net_grams[i]):
                    string=string+grams[j]+" ";
               try:
                    if mydictionary[string]>=0:
                         flag=1;
               except:
                    flag=0;
               if flag==1:
                    ind=mydictionary[string];
                    count[ind]=count[ind]+1;
          for frequency in range(len(count)):
               temp7=str(count[frequency])+","
               fw[i].write(temp7)
          temp7=filetype+"\n\n"
          fw[i].write(temp7)
          #os.chdir(cwd1)
for t in range(len(all_net_grams)):
     temp="mv "+str(all_net_grams[t])
     temp=temp+"gramsfrequency "+cwd0+"/output/Training_Data_Master";
     os.system(temp)
```

Recursing over all the files in Training_Data_Master, it reads all the files and finds its ngrams and store it in single_grams. Then iterating over all those grams, the code counts the frequency of the features as found out and writes them in a file. Finally, all the files as produced is sent to the output folder and within it it goes to Training Data Master folder

THE END