



INDIAN INSTITUTE OF INFORMATION TECHNOLOGY
ALLAHABAD

5TH SEMESTER MINI PROJECT

Side Channel Attack

Submitted To:

Dr. Bibhas Ghoshal

Submitted By:

Anurag Bhardwaj (IIT2015076)

Kautish Jaiswal (IIT2015506)

Nitesh Gupta (IIT2015050)

Ayan Sheikh (IIT2015079)

Yash Bhatia (IIT2015105)

CERTIFICATE FROM SUPERVISOR

I do hereby recommend that the mini project report prepared under my supervision, titled “**Side Channel Attack**” be accepted in the partial fulfillment of the requirements of the completion of mid-semester of fifth semester of Bachelor of Technology in Information Technology.

Date:

Place: Allahabad

Supervisor :

.....

Dr. Bibhas Ghoshal

Abstract

Information technology (IT) is evolving everyday and our day-to-day life is becoming more and more dependent on it. We cannot imagine ourselves without online banking systems, social networking sites, video streaming and World Wide Web. Due to this evolution of Internet our life has become more convenient and technologically advanced. During this course we are exploring the dark side and horrors of Internet regarding hacking and cyber-crimes. Apart from these direct attacks, side channel leaks can be used on search engines like Google, Bing, Amazon to retrieve clients' sensitive information. There is a wrong assumption that encryption restricts data access to only authorized users. Side Channel leaks are the direct results of implementation constraints giving access of clients' data to unauthorized users.

We explore the practicality of one such side-channel attack where the attacker deduces what was typed in the search suggest box (suggestions to partially typed query) by just observing the sizes of the packets exchanged between a user and an access point. In order to uniquely map a search query to a web traffic signature, we make use of Stochastic Algorithms.

Our goal is to find an optimal method to implement the side-channel attack using the packet lengths of data. We also present some methods to mitigate such side-channel attacks.

Contents

Certificate	i
Abstract	ii
1 Introduction	1
2 Literature review and Motivation	2
3 Problem Definition	3
4 Methodology and Implementation	5
4.1 Reduction of possibilities	5
4.2 Data Structure	5
4.3 Packet Sniffing	6
4.4 Observation on Google packets	7
4.5 Observations on Bing packets	7
4.6 Dataset Generation	9
4.7 Data Analysis	9
4.8 Algorithms	11
4.8.1 Algorithm to find the best match	11
4.8.2 Algorithm for suggestion	12
4.9 ARP Poisoning	14
5 Results and Evaluation	16
6 Conclusion	23
6.1 How to Mitigate Side Channel Attack	23

1 Introduction

Web applications nowadays are vulnerable to number of attacks. SQL injection, cross-site scripting (XSS) attacks and cross-site request forgery (CSRF) are some of the biggest threats on web applications as reported by the Open Web Application Security Project[1].

The sensitive information exchanged between two entities is encrypted to ensure that they are not eavesdropped upon by unintended parties. Even with such security mechanisms in place, some information get leaked by the system unintentionally. This leaked information can be catastrophic enough for the system.

A side channel has been determined as a medium or an entity from where any observable information is emitted as a byproduct of the physical implementation of any system OR any information not captured by the abstract standard model of a system. Side channel attacks have been studied in various contexts in last two decades, the information leakage might be through the electromagnetic radiations emanated by a keyboard, heat dissipation by the hardware, power usage by the processor, shared memory/files between processes, size of data exchanged and others. Even in a secured encryption setup some side-channel information can be used to get information about the communication between the two entities[2].

In our project, we address how to exploit a side-channel made by a web-browser. We predict the queries entered by a victim in the AJAX based search boxes by examining the packets sent in a secured WiFi network even when packets are encrypted.

2 Literature review and Motivation

In cryptography, a side-channel attack is any attack based on information gained from the physical implementation of a crypto-system, rather than brute force or theoretical weaknesses in the algorithms[3].

Numerous studies on the detection and analysis of side-channel data leaks in web applications can be found in the literature. The general approach is to examine the properties of packet sequences sent between a client and a server in order to infer a relationship between these properties and the exchanged information.

Online security breaching is a criminal activity that can happen from any part of the world. Most breaches are done by falsifying the personal information or even a breach of security stemming from lack of security. Now that more and more crimes are happening over the Internet there are many different aspects to security as a whole. Due to the drastic increase in modern cyber crime the cyber police, information technology specialists, and even the government need to work together to find a solution.

A data breach is a security incident in which sensitive, protected or confidential data is copied, transmitted, viewed, stolen or used by an individual unauthorized to do so[4].

Many times private organizations like Wikileaks have exposed secret military documents and unveiled government policies[5].

Until 2011, Google searches were not encrypted. The search giant enabled encryption for its signed in users[6] in stages and quickly followed it by a complete roll-out[7] of its SSL based version in late 2012.

There are many incidents of data breaches in the history of Internet[8]. This motivated us in following ways:

1. Gaining access to a client's search history has huge privacy implications, and privacy is intimately related to computer security.
2. Search history is also of commercial importance, with companies using it for targeted user interest based advertisements.

Recent studies show that such side-channel leaks are both fundamental and realistic. A set of popular web applications are found to disclose highly sensitive user data such as one's family income, health profiles, investment secrets and much more through their side channels. Our study also shows that a significant improvement of the current web-application development practice is necessary to mitigate this threat.

3 Problem Definition

Information leaked in web applications can be used to gain sensitive information like what a user typed in a search box. Search engines like Google, Bing provide an auto complete suggest box to assist the user for every character keyed in as input.

Even if the connection between the client and server is encrypted, three parameters of packet flow can be observed for HTTPS protocol:

1. Lengths of individual packets.
2. Directions of packet flow (client to server or server to client).
3. Times of packets' departure and arrival.

It is observed that for every character typed in the search box, several packets are exchanged between the server and the client with list of suggestions:

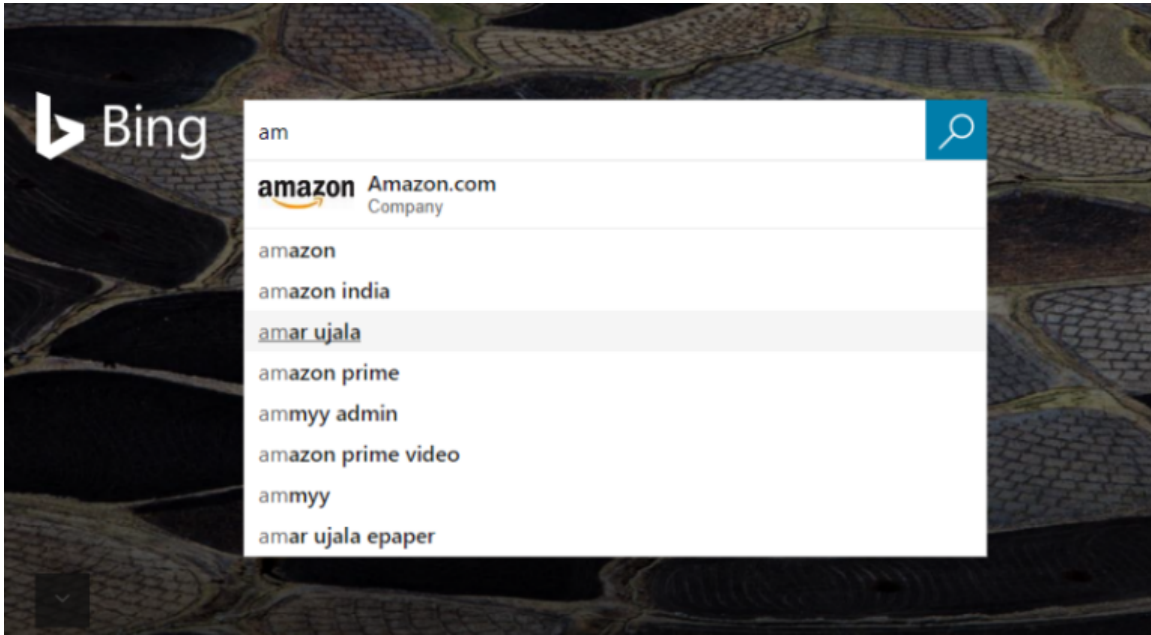


Figure 1: Search suggestions from Bing

The search box operates using AJAX to display suggestions. We can attack by intercepting the exchanged packets in order to infer the users' query.

Exchanged packets between a client and server can be observed using a packet sniffer such as *Wireshark*. Captured packets are read by *Pyshark*. We maintain the records of packet lengths observed when each word in the dictionary is typed as a search query.

We check the functionality of side-channel attacks in a secured WiFi setup where an attacker (an unauthenticated user) can predict what a victim user searched on

another machine without validation. For every character keyed in the search box by the victim, the packet length follows a fixed pattern. These were analyzed to predict what the victim searched.

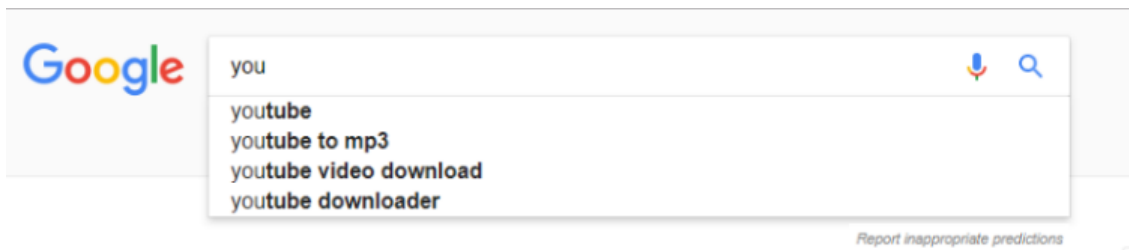


Figure 2: Search suggestions from Google

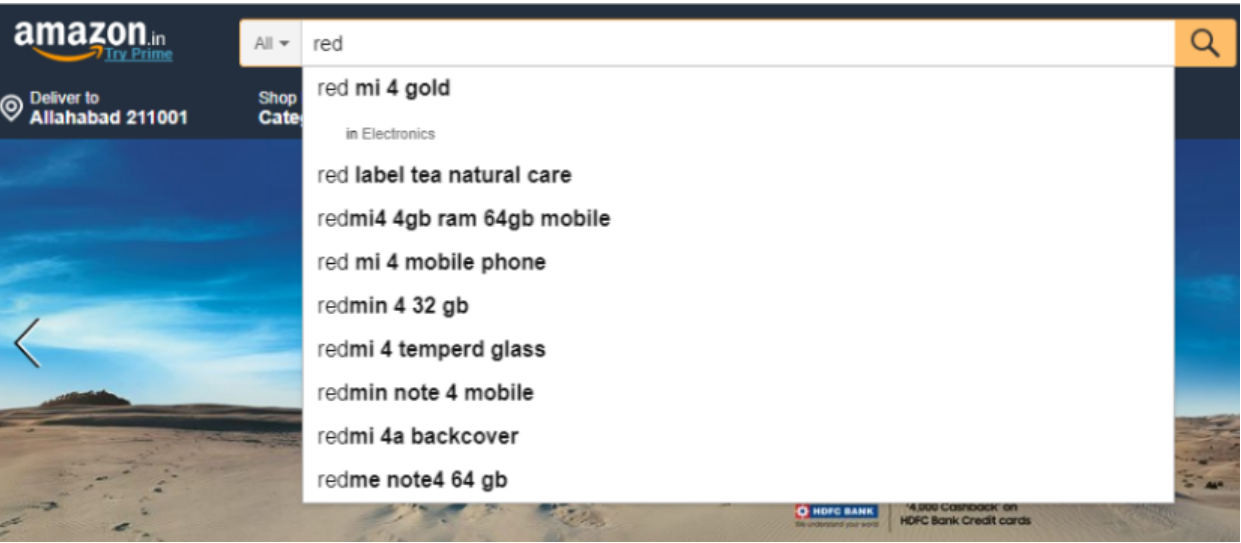


Figure 3: Search suggestions from Amazon

4 Methodology and Implementation

From the above mentioned three packet parameters, packet length and direction can be used to guess the key stroke.

4.1 Reduction of possibilities

To find a string of size n the victim of our attack has to send 26^n automated search requests to the server. We observed that most of the search requests do not qualify to be a word that one might search. This process can be optimized by eliminating a lot of search requests and responses as they may not be legitimate English words and thus, restricting it to a dictionary, i.e. set of predefined words.

4.2 Data Structure

The packet intercepted from the search suggestions represent prefix of the word. Thus the best data structure suitable to deal with prefix string matching i.e. Trie tree is used. Once the dictionary is chosen, we construct a trie tree which stores all the meaningful words of a dictionary along with probability ranking based on various search engines query dataset. Each node has 26 children each representing an alphabet with the root node representing the *null* character. Likewise subsequently each level l represents valid words of length l sorted lexicographically. The below snippet describes the trie further.

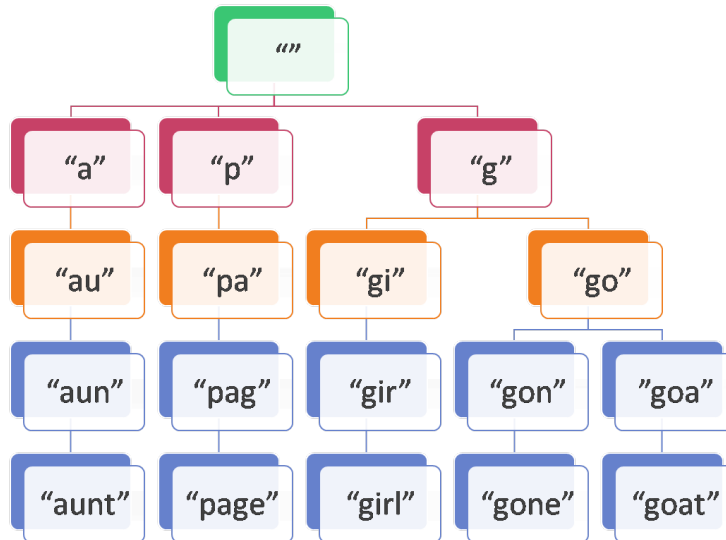


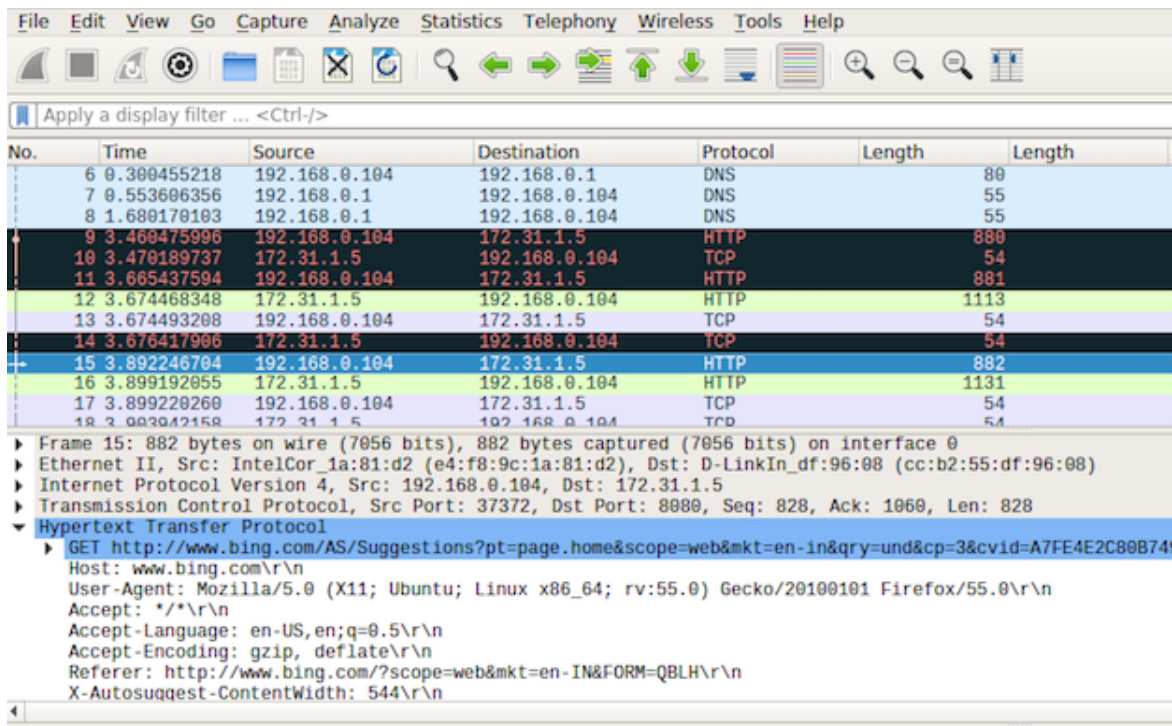
Figure 4: Structure of Trie

4.3 Packet Sniffing

a) **Through Wireshark:** Wireshark is the world's foremost and widely-used network protocol analyzer. It lets you see what's happening on your network at a microscopic level and is the de facto (and often de jure) standard across many commercial and non-profit enterprises, government agencies, and educational institutions[9].

Packets corresponding to search suggestions are captured using packet sniffers such as Wireshark using suitable filters.

Wireshark needs the interface to receive all the packets that it sees whether they are addressed to it or not. Hence, we run wireshark in promiscuous mode.



No.	Time	Source	Destination	Protocol	Length	Length
6	0.300455218	192.168.0.104	192.168.0.1	DNS	80	
7	0.553606356	192.168.0.1	192.168.0.104	DNS	55	
8	1.680170103	192.168.0.1	192.168.0.104	DNS	55	
9	3.460475996	192.168.0.104	172.31.1.5	HTTP	880	
10	3.470189737	172.31.1.5	192.168.0.104	TCP	54	
11	3.665437594	192.168.0.104	172.31.1.5	HTTP	881	
12	3.674468348	172.31.1.5	192.168.0.104	HTTP	1113	
13	3.674493208	192.168.0.104	172.31.1.5	TCP	54	
14	3.676417906	172.31.1.5	192.168.0.104	TCP	54	
15	3.892246704	192.168.0.104	172.31.1.5	HTTP	882	
16	3.899192055	172.31.1.5	192.168.0.104	HTTP	1131	
17	3.899220260	192.168.0.104	172.31.1.5	TCP	54	
18	3.003042158	172.31.1.5	192.168.0.104	TCP	54	

Frame 15: 882 bytes on wire (7056 bits), 882 bytes captured (7056 bits) on interface 0
 Ethernet II, Src: IntelCor_1a:81:d2 (e4:f8:9c:1a:81:d2), Dst: D-LinkIn_df:96:08 (cc:b2:55:df:96:08)
 Internet Protocol Version 4, Src: 192.168.0.104, Dst: 172.31.1.5
 Transmission Control Protocol, Src Port: 37372, Dst Port: 8080, Seq: 828, Ack: 1060, Len: 828
 Hypertext Transfer Protocol
 GET http://www.bing.com/AS/Suggestions?pt=page.home&scope=web&mkt=en-in&qry=und&cp=3&cvid=A7FE4E2C80B74
 Host: www.bing.com\r\n
 User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:55.0) Gecko/20100101 Firefox/55.0\r\n
 Accept: */*\r\n
 Accept-Language: en-US,en;q=0.5\r\n
 Accept-Encoding: gzip, deflate\r\n
 Referer: http://www.bing.com/?scope=web&mkt=en-IN&FORM=QBLH\r\n
 X-Autosuggest-ContentWidth: 544\r\n

Figure 5: Capturing packet using Wireshark

The above snippet shows how wireshark is used to predict the length of the search suggestion packet.

b) **Through Pyshark:** Python wrapper for tshark, allowing python packet parsing using wireshark dissectors. We use pyshark library to capture live packets through a simple python code[10]. Packet lengths are captured for various possible keystrokes and recorded in the dataset. The following snippet shows how to read packets using pyshark:

Reading from a capture file:

```
>>> import pyshark
>>> cap = pyshark.FileCapture('/tmp/mycapture.cap')
>>> cap
<FileCapture /tmp/mycapture.cap (589 packets)>
>>> print cap[0]
Packet (Length: 698)
Layer ETH:
    Destination: BLANKED
    Source: BLANKED
    Type: IP (0x0800)
Layer IP:
    Version: 4
    Header Length: 20 bytes
    Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT (Not ECN-Capable Transp
    Total Length: 684
    Identification: 0x254f (9551)
    Flags: 0x00
    Fragment offset: 0
    Time to live: 1
    Protocol: UDP (17)
    Header checksum: 0xe148 [correct]
    Source: BLANKED
    Destination: BLANKED
...

```

Figure 6: Reading packets from pcap file using pyshark

Search Engines use AJAX technology to provide search suggestions. Every time a letter is typed in the search box, a request is sent from client browser to the server requesting it to predict what the user might be typing and provide suggestions accordingly. These requests are nothing but packets exchanged between server and the browser.

4.4 Observation on Google packets

For any letter typed in the Google search box, the packets exchanged were difficult to trace as Google sends a large number of irrelevant packets in order to increase the randomness. Further, length of packets varied rapidly for same character tested more than once. For example- when tested for character 'a' the packet size varied from 67-1250. Thus, no conclusion could be drawn from the packet lengths.

4.5 Observations on Bing packets

When the character 'a' was typed in the search box of the bing, a packet of length 913 was sent. Upon further typing and analyzing other characters it can be easily seen that the length of the packets sent were constant, i.e. 913, at least for the same

session. Further it was noticed that when second character was typed the size of the second packet being sent incremented by one, i.e. 914, irrespective of the characters. Thus, the sent packets can be used to determine the number of keystrokes made which was correct in approximately **96%** of the cases.

Still the question of guessing the keystrokes from the packet length remain unsolved. The answer lies in the suggestion packets sent by the server as a response to the request made by client. It was observed that the length of the packets received varied from alphabet to alphabet. Further the packets had a fixed part of 716 bytes(for a particular session) which comprise of various HTTP headers irrespective of requests. Thus,

$$\text{Request length} - \text{Header length} = \text{Suggestion Length}$$

Moreover the suggestion length was nearly same for a keystroke in **67%** of the cases. The table below shows our observations for some strings after multiple attempts:

String	Packet Length Sent	Packet Length Received(Total)	Suggestion Packet Length	Header Length
a	913	1254	538	716
a	913	1253	537	716
b	913	1260	544	716
b	913	1261	545	716
c	913	1264	548	716
c	913	1263	547	716
ab	914	1238	522	716
ac	914	1234	521	716
ac	914	1238	522	716
ad	914	1265	549	716
ad	914	1265	549	716
ae	914	1230	514	716
ba	914	1283	567	716

Table 1: Packet Analysis Table

Based on above observations we have implemented a python code which uses pyshark library to sniff live packets between browser and server. Again using packet lengths from the server side and mapping them with the observation table we are able to guess the string that was entered by user to some extent (for strings upto length 3).

4.6 Dataset Generation

An important step in the project was to create the database. We bifurcated our dataset into 3 parts, a file for 1 character prefix, 2 characters prefix and 3 characters prefix each. The dataset includes prefixes of valid dictionary words only so as to reduce the possible permutations.

Initially we typed each prefix manually in bing to capture packets. This procedure was time consuming as permutations are quite high. To overcome this, a simple JavaScript code (Figure 7) is used which takes input from a file word by word and writes it in the bing search box after a fixed amount of time interval[11].

```
1  var i = 0;
2  var lineArr;
3  function reader()
4  {
5      document.getElementById("sb_form_q").value = lineArr[i];
6      console.log(lineArr[i]);
7      i++;
8  }
9
10
11 function readTextFile(file)
12 {
13     var rawFile = new XMLHttpRequest();
14     rawFile.open("GET", file, false);
15     rawFile.onreadystatechange = function ()
16     {
17         if(rawFile.readyState === 4)
18         {
19             if(rawFile.status === 200 || rawFile.status == 0)
20             {
21                 var allText = rawFile.responseText;
22                 lineArr = allText.split('\n');
23                 setInterval(reader, 5000);
24                 alert(allText);
25             }
26         }
27     }
28     rawFile.send(null);
29 }
```

Figure 7: Javascript to write data in search box

4.7 Data Analysis

Orange is a platform built for mining and analysis on a GUI based work flow[12]. Orange, a data mining software, takes the data set generated by us as an input in *.tab* format and correspondingly generates the classification tree as an output. Classification tree is control flow graph for our pre-built dataset which categorizes keys according to values.

Basis of our project is to predict the word given packet length and this prediction process is achieved through Orange. Given a data length, Orange predicts an appropriate word using its inbuilt prediction algorithm. This toolkit uses common open source libraries written in python in computing and predicting the key for a given value i.e. packet length. A popular supervised machine learning is being used in this software for the prediction process. The generated dataset was analyzed and the classification tree was observed.

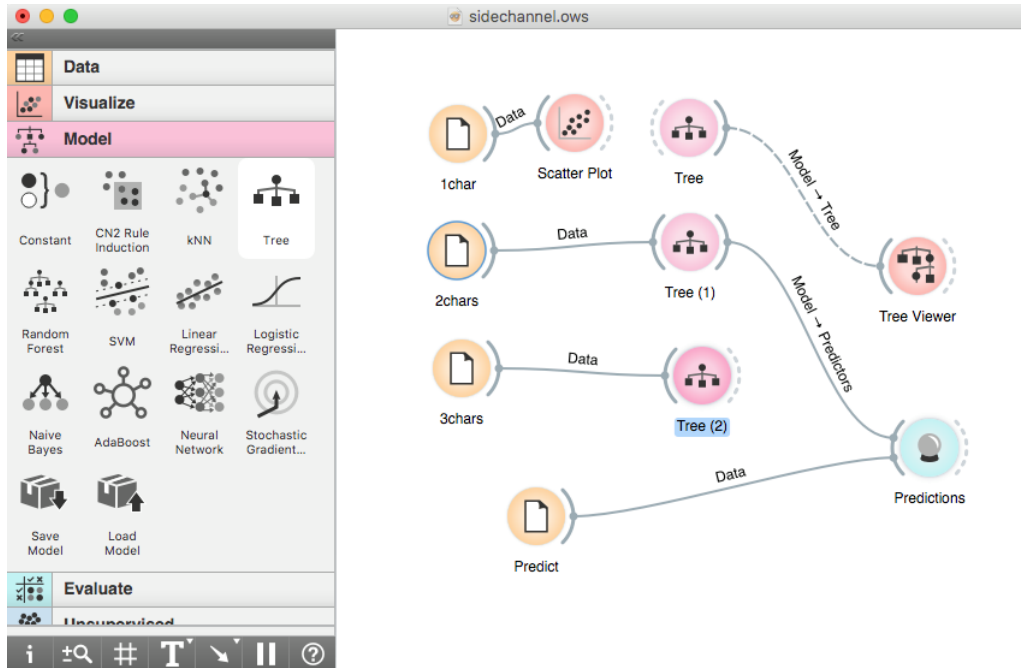


Figure 8: Creation of Classification tree

Implementation using Orange results in an accurate prediction of a single letter with accuracy to almost **95%**. But unfortunately this gradient becomes negative when it comes to words containing more than one letter. We observed that as the number of letters in the word increased the accuracy decreases significantly. This made us switch to another approach.

4.8 Algorithms

We used two algorithms in our projects:

- a) To find the best match.
- b) The possible suggestions for that match.

4.8.1 Algorithm to find the best match

We have 3 packet lengths $L1$, $L2$, $L3$ received on typing 1st, 2nd and 3rd character respectively. An approximate value of packet length for each string is calculated by taking average over all the values stored in the dataset. The mapping calculated is stored in distinct files for distinct length.

We consider 3 tolerance level ϵ_1 , ϵ_2 and ϵ_3 for length of string 1, 2 and 3 respectively. Now for calculating best match, we found all those strings of length 1 such that its mapped value($V1$) lies between $[L1 - \epsilon_1, L1 + \epsilon_1]$. We stored these strings into another map $M1$ with mapped value as δ_1 (difference between $V1$ and $L1$).

Similarly we found all those string of length 2 such that their mapped length($V2$) lies in range $[L2 - \epsilon_2, L2 + \epsilon_2]$ and also their prefix lies in $M1$. Again we stored these strings into map $M2$ with mapped value as δ_2 ($\delta_1 +$ difference between $V2$ and $L2$). The same step is repeated for strings of length 3 but instead of map, the pair of δ_3 and string is stored in a set sorted according to δ_3 . Whenever size of set exceeds 10 pop the element with maximum δ_3 ($\delta_2 +$ difference between $V2$ and $L2$).

```

Create a map m(string to integer)

Read data from DATA1(s, val)
    diff1 = abs(val - L1)
    if diff1 < 10:
        m[s] = diff1

create a map m2(string to integer)

Read data from DATA2(s, val)

    str = s.substring(0, 1)
    diff2 = abs(val - L2)
    if str in m and diff2 < 10:
        m2[s] = m1[str] + diff2

create a set of pair(integer, string) st

Read data from DATA3(s, val)
    diff3 = abs(val - L3)
    str = s.substring(0, 2)
    if str in m2 and diff3 < 10:
        st.insert({m2[str] + diff3, s})
        if st.size() > 10:
            delete the data containg maximum summation of delta from set st

iterate through whole set(val, s) in increasing order of val
    print s
    print val

```

Figure 9: Algorithm to find the best match

4.8.2 Algorithm for suggestion

We construct the trie tree of all the words in dictionary with priority as its position in file. Now from the previous algorithm we extract 3 strings with minimum δ_3 and find the best suggestions for those strings using trie tree.

Construction of Trie Tree: We focus on valid English words only so we maintain a trie tree with 26 child rather than whole character set. To increase the speed we implemented the trie with array, as pointers are slightly slow.


```

ctr = 1
root = 1
Read data from Dictionary(word, priority)
trav = root
for i in range word.size()
    nx = word[i] - 'a'
    if tree[trav].ch[nx] == NULL
        tree[trav].ch[nx] = ++ctr
        trav = tree[trav].ch[nx]

tree[trav].pr = priority
tree[trav].isLeaf = True

```

Figure 10: Algorithm to construct the trie tree

Finding the suggestion: Now suppose we have a string s and we want to find best suggestion for that string from our limited dictionary words.

```

trav = root
for i in s.size()
    nx = s[i] - 'a'
    trav = tree[trav].ch[nx]
findBest(trav, s)

def findBest(trav):
    create an empty set st
    findRec(st, trav, s)
    print set st

def findRec(st, trav, s):
    if tree[trav].isLeaf == true
        st.insert((tree[trav].pr, s))
        if st.size() > 4:
            remove element with maximum value of priority

    for i in range 26:
        if tree[trav].ch[i] != NULL:
            findRec(st, tree[trav].ch[i], s + (char)(i + 'a'))

```

Figure 11: Algorithm to find the suggestion for the best match

4.9 ARP Poisoning

ARP spoofing is a type of attack in which a malicious actor sends falsified ARP (Address Resolution Protocol) messages over a local area network. This results in the linking of an attacker's MAC address with the IP address of a legitimate computer or server on the network. Once the attacker's MAC address is connected to an authentic IP address, the attacker will begin receiving any data that is intended for that IP address. ARP spoofing can enable malicious parties to intercept, modify or even stop data in-transit. ARP spoofing attacks can only occur on local area networks that utilize the Address Resolution Protocol.

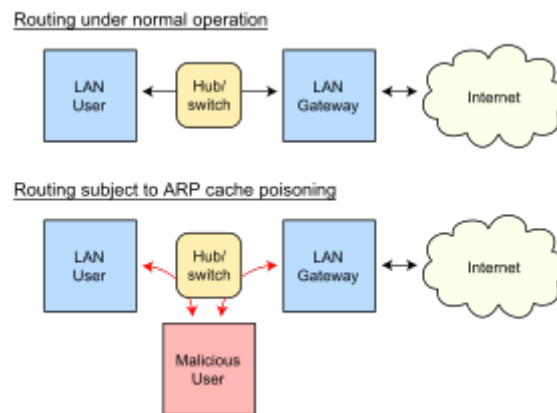


Figure 12: ARP poisoning

We have implemented this ARP spoofing using Ettercap toolkit[13].

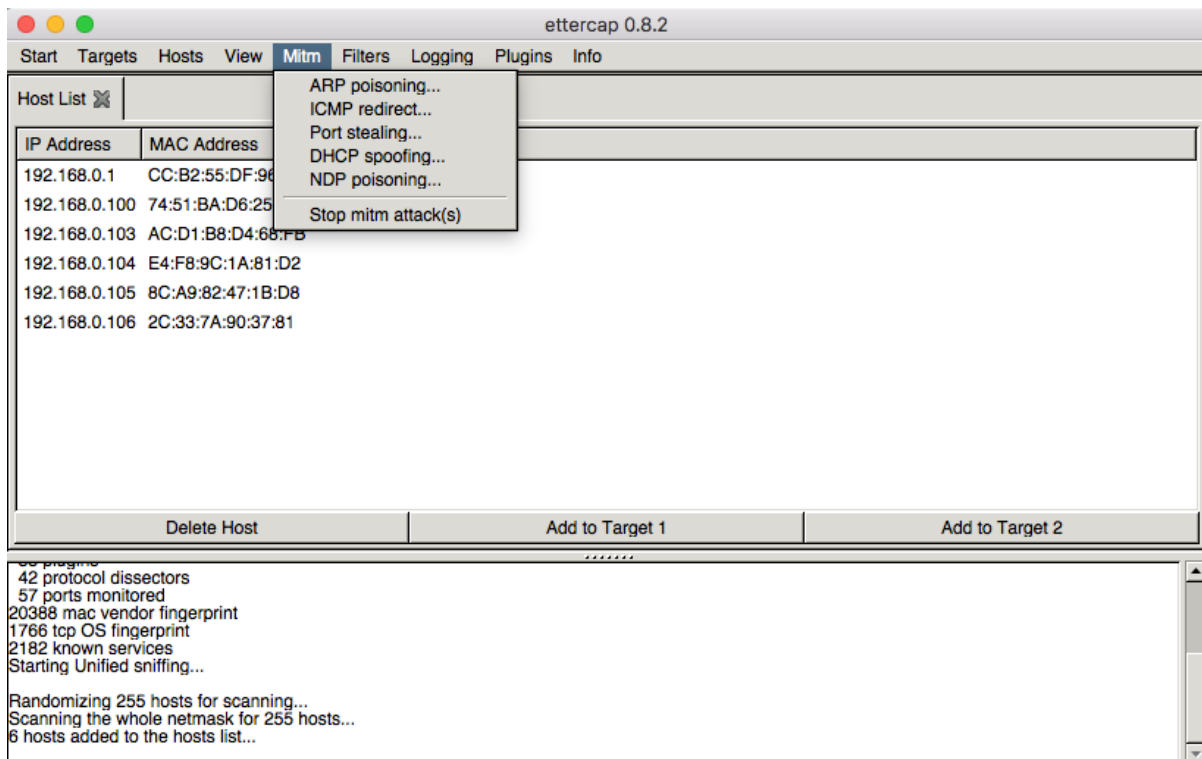


Figure 13: MITM using Ettercap

5 Results and Evaluation

This section presents the results of our algorithm tested on bing.com, by simulating an interception over Ethernet or Wifi. As the space and time complexity of the algorithm increases with the length of the query string, we assumed a 3 letter word is typed by the user.

Observations using orange: The dataset created by simple JavaScript and AJAX query was saved in the orange specified input format separated by tabs in *.tab* file and analyzed.

Packet Lengths: The data was passed through distribution graph plot in orange. It was observed that all the suggestion packets lie within 0-2000 range. The exponential growth in the region 450-600 suggest that the highest density of packets lie in this region. The following graph shows the above observation.

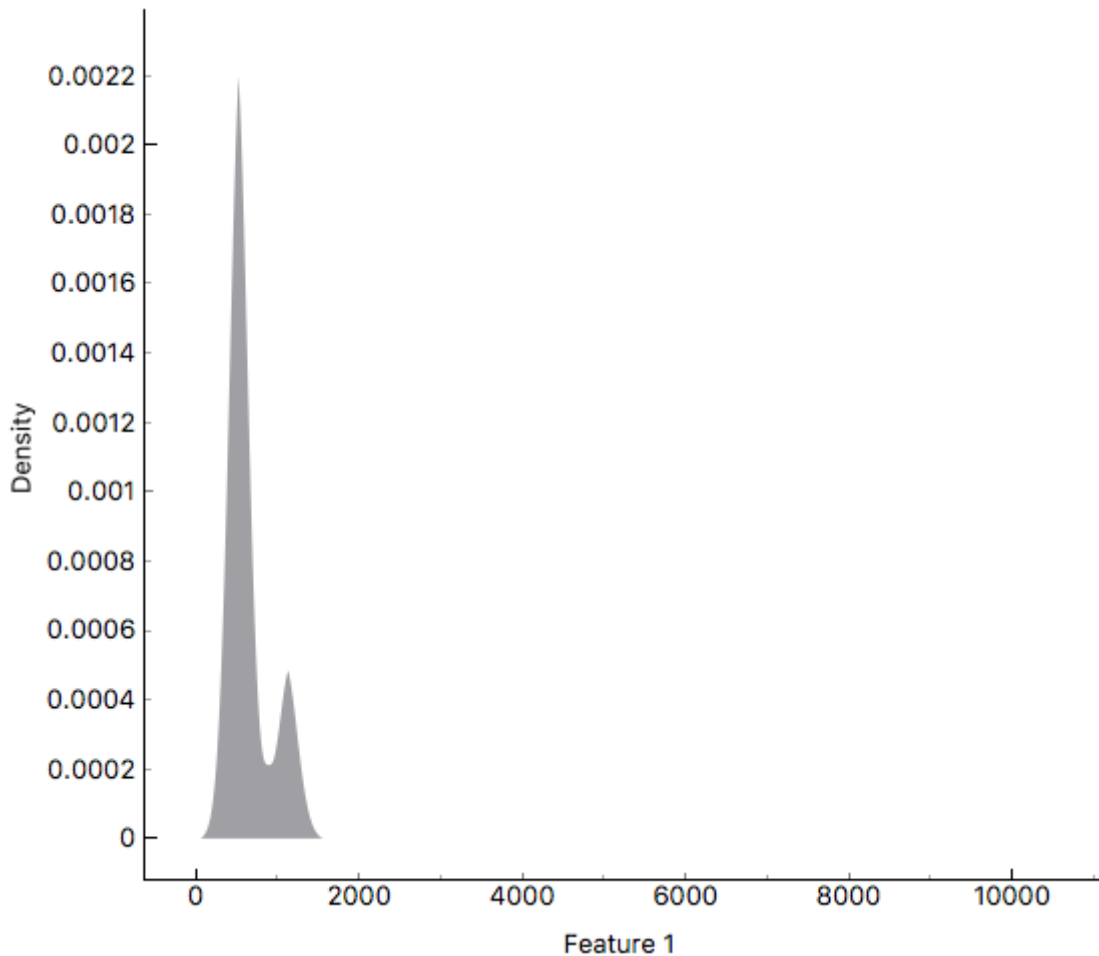


Figure 14: Distribution Graph

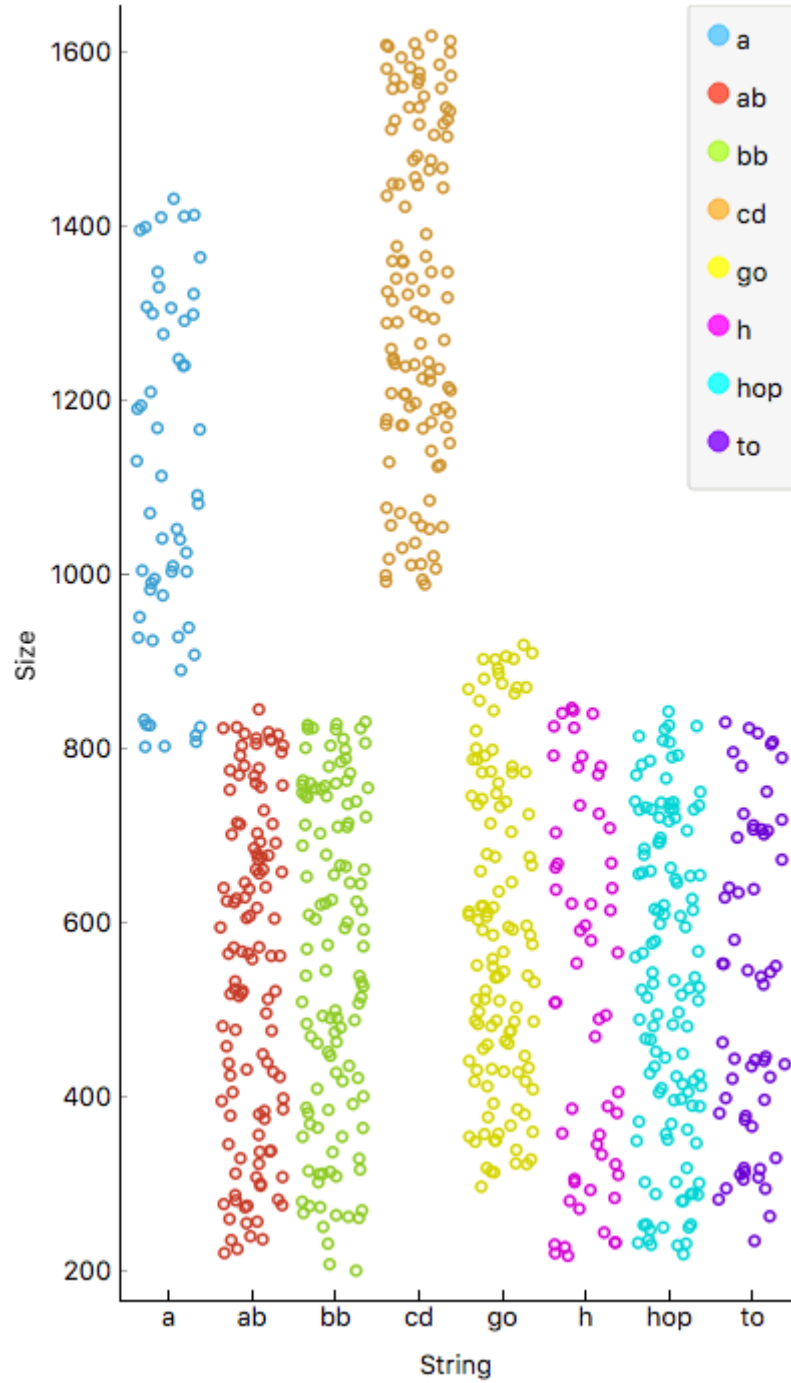


Figure 15: Scatter Plot

We could easily map queries of length one and apply orange classification algorithm based on their packet lengths for predictions. The following classification tree

summarizes the above discussion.

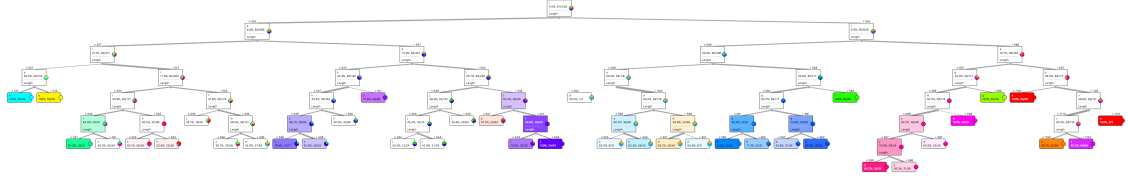


Figure 16: Classification tree for 1st character

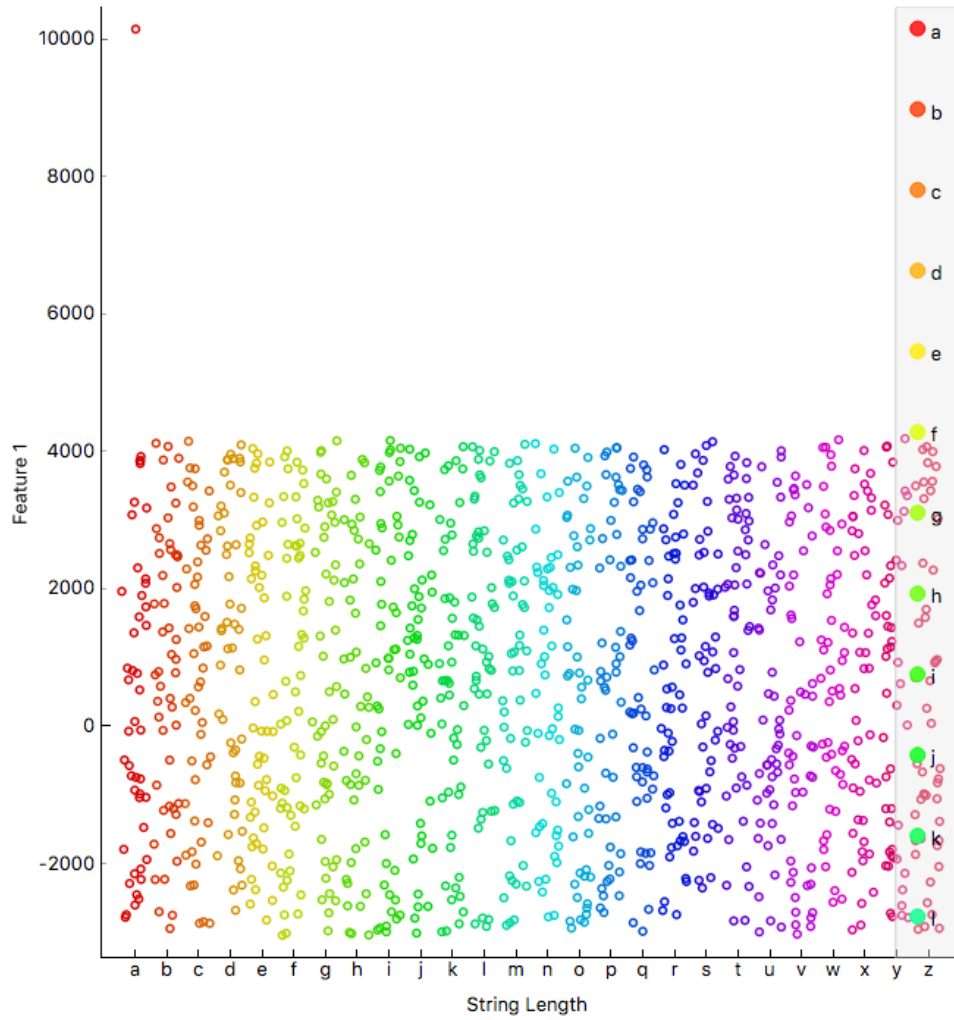


Figure 17: Distribution Graph

Further the graph (figure 13) suggest that most of the words lie in the 150(600-450) length variations. As a result one can conclude that it is not possible to map

strings of length more than 2 to their respective packet lengths easily.

Observation using our algorithm: The number of stored prefixes for each length (1 to 3) were chosen based on minimum delta value about the mean. The final list thus contains ten most probable strings ranked on the basis of their delta values.

The code was tested for 100 queries. We checked the result for top 10 strings and the topmost string. The success rate that the string was in the final list and at the top was **96%** and **80%** respectively. The following tables show some of the observations:

String	In Top 10	In Top 1
app	Yes	No
del	Yes	Yes
esc	Yes	Yes
cha	Yes	Yes
hom	Yes	Yes
gre	Yes	Yes
ato	Yes	Yes
rea	Yes	Yes
tex	Yes	Yes
jem	No	No
you	Yes	Yes
hap	No	No
ret	Yes	No
onl	Yes	Yes
hat	Yes	Yes
rob	Yes	Yes
des	Yes	Yes
hom	Yes	Yes
sub	Yes	Yes

Table 2: Result Verification Table

Similarly we tried to attack other system on the same network using the concepts of ARP poisoning and obtained almost comparable results.

String	In Top 10	In Top 1
tex	Yes	Yes
pol	Yes	Yes
gue	Yes	Yes
lam	Yes	Yes
bla	Yes	No
num	Yes	Yes
ven	Yes	Yes
for	Yes	No
cha	Yes	Yes
dum	Yes	Yes
nip	Yes	Yes
wet	Yes	Yes
dar	Yes	No
ore	Yes	Yes
rus	Yes	Yes
zip	Yes	Yes
bac	Yes	Yes
pom	No	No
lat	Yes	Yes
cra	Yes	Yes

Table 3: Result Verification Table (for third party system)

For calculating the accuracy of our algorithms the code was tested for 10 target strings with 10 re-tries per target. The table 4 shows the rank $\epsilon \in [1,10]$ of the target word in the final list.

Word	N1	N2	N3	N4	N5	N6	N7	N8	N9	N10
cri	1	1	1	1	1	1	1	1	1	1
gov	1	1	1	1	1	1	1	1	1	1
fuc	2	1	2	1	2	1	1	1	2	1
mea	1	1	1	1	1	1	1	1	1	1
sel	1	1	1	1	4	1	1	2	1	1
tex	1	2	1	1	1	1	2	1	1	1
arg	1	1	1	1	1	1	1	1	1	1
aar	2	1	2	2	1	1	2	1	1	1
sep	1	1	2	2	1	1	1	1	2	2
jee	1	1	1	1	1	1	1	1	1	1
dis	2	1	1	2	2	1	1	2	1	2
upl	1	1	1	1	1	1	2	1	1	1

Table 4: Suggestion Analysis Table

Out of 120 attempts, 98 times the suggestion came at the top. So the accuracy for the experimented data is approximately **82%**. However for the suggestion being in top 3 the accuracy is nearly **99%**.

```
Niteshs-MacBook-Pro-2:sidechannel Nitz$ sudo python findBestMatch_final.py
SENT
[1
RECEIVED
550
SENT
2
RECEIVED
544
SENT
3
RECEIVED
612
BREAKING..
(2, 'din')
(2, 'dis')
(3, 'dee')
(7, 'did')
(11, 'den')
(11, 'cac')
(13, 'htt')
(13, 'joi')
(14, 'dri')
(14, 'job')
Suggestion for 1st word
dining
dinner
Suggestion for 2nd word
dis
disabilities
disability
disable
Suggestion for 3rd word
dee
deemed
deep
deeper
```

Figure 18: Final Result

6 Conclusion

In our project we analyzed the side-channel leakage of Bing browser. We used knowledge of packet lengths to find out user's search query. We argued that by just looking at packet lengths one can find what a victim was searching despite working in a secure WiFi network. An optimum algorithm is used to accomplish this based on difference in packet lengths for each typed prefix. Our algorithm can be modified to any other search engine which uses AJAX based search boxes.

Some improvements and issues remain topics for future investigations.

a) **Use of Backspace Key:** The algorithm cannot predict the search query if a user hits the backspace key as it would be searching for a word that doesn't exist in our dataset. For instance the word "sun" -> "sub", the new word would look like "sun(<-)b" which is not in our dataset. Further it would be looking for a word that will be too long. For the word, "sun(<-)b", one would receive 5 packets related to queries "s", "su", "sun" , "su" and "sub".

b) **Use of Multiple Words:** Our code doesn't work efficiently if query consisting of more than one word is searched. To incorporate the use of the space key, it would be required to slightly change the structure of the trie tree. Every leaf (word) should be arrowed back to the root, where the arrow represents the whitespace character. It will look like a cyclic structure instead of a tree.

c) **Wait for Acknowledgment:** In our project we need to wait for a considerable amount of time after typing every character to capture packets correctly. This is because the server response time is slow on the network. We need to wait for acknowledgment to come from the server. The better the response time of the server, faster we could type.

d) **Limited Dictionary:** Our current project restricts itself to a small domain with limited number of words. We can improve it by making a exhaustive dictionary which includes almost all valid words in English. Also, we can extend our project in different language like French by making a dictionary containing authentic words from that language and updating Trie structure.

6.1 How to Mitigate Side Channel Attack

As we have shown, it is possible to observe on anybody using a Wifi connection, even if this is secure connection. This is a serious threat to privacy over the Internet. A web application is split between the browser and the server, so the data is exchanged through the network. Despite everything, some elementary characteristics of web applications make the side-channel leak a practical and severe privacy problem[14]. There have been many attempts to reduce side-channel leaks, but it is generally considered that prohibiting every side-channel leak source is very hard.

However, for the particular leak exploited in our project, it would be simple to implement a powerful counter-measure by randomizing the packet lengths. Randomization of packet lengths makes it difficult to conclude a search query. Every time a victim types a query the length of the packet exchanged will be random. Thus the attacker cannot map a certain packet length to a string. It would require stochastic approach and improved algorithms to deduce the string in such case.

References

- [1] E. Ferdkin, “Owasp.” <http://doi.acm.org/10.1145/367390.367400>, 1960. [Online; accessed November 2017].
- [2] S. Sharma, “Implementing side channel attacks.” http://delivery.acm.org/10.1145/2500000/2490436/p57-sharma.pdf?ip=14.139.236.210&id=2490436&acc=ACTIVE%20SERVICE&key=045416EF4DDA69D9%2EF1486EDFD37746FC%2E4D4702B0C3E38B35%2E4D4702B0C3E38B35&CFID=831597725&CFTOKEN=61543992&__acm__=1511172697_3b330fcf5770dfb287b63e493b14c375, 2012. [Online; accessed November 2017].
- [3] wikipedia, “Side channel attack.” https://en.wikipedia.org/wiki/Side-channel_attack, 2017. [Online; accessed November 2017].
- [4] wikipedia, “Data breach - a threat.” https://en.wikipedia.org/wiki/Data_breach, 2016. [Online; accessed September 2016].
- [5] C. Joseph, “Wikileaks releases secret report on military equipment.” <http://www.nysun.com/foreign/wikileaks-releases-secret-report-on-military/62236/>, 2007. [Online; accessed September 2017].
- [6] E. Kao, “Making search more secure.” <https://googleblog.blogspot.in/2011/10/making-search-more-secure.html>, 2011. [Online; accessed November 2016].
- [7] D. Sullivan, “Post-prism, google confirms quietly moving to make all searches secure, except for ad clicks.” <http://searchengineland.com/post-prism-google-secure-searches-172487>, 2013. [Online; accessed November 2016].
- [8] T. Armerding, “16 biggest data breaches of 21st century.” <https://www.csoonline.com/article/2130877/data-breach/the-16-biggest-data-breaches-of-the-21st-century.html>, 2017. [Online; accessed September 2017].
- [9] “Wireshark - about.” <https://www.wireshark.org/>, 2017. [Online; accessed September 2017].
- [10] “Pyshark - about.” <https://thepacketgeek.com/intro-to-pyshark-for-programmatic-packet-analysis/>, 2014. [Online; accessed November 2017].

- [11] “Javascript - about.” <https://www.javascript.com/learn/>, 2017. [Online; accessed November 2017].
- [12] “Orange - about.” <https://blog.biolab.si/>, 2017. [Online; accessed November 2017].
- [13] “Ettercap - about.” <https://pentestmag.com/ettercap-tutorial-for-windows/>, 2017. [Online; accessed November 2017].
- [14] S. Chen, “Side channel leak in web operations.” <https://people.eecs.berkeley.edu/~raluca/cs261-f15/readings/webchannel.pdf>, 2010. [Online; accessed November 2017].