

# Side Channel Attack on Search boxes of Web Applications

Group members -

Pallak Srivastava (IIT2016002)

Mansi Choudhary (IIT2016016)

Shubhangi Nigam (IIT2016023)

Simran Gill (IIT2016088)

Sudhanshu (ISM2016004)

Project Supervisor -

Dr. Bibhas Ghoshal



---

# Outline

- The Problem Statement
- Motivation
- Approach
- Next Steps
- Final Analysis



---

# Problem statement

Attacking Search boxes of Web applications via Side Channel Attacks in order to intercept and analyse the query typed by user .



---

# Motivation

- High level of data encryption is also susceptible to data leaks due to its physical implementation which possess a threat to all data transfers.
- A data transfer whose physical implementation is highly random tends to be more secure.
- But still there are limitations to such implementations.
- Our motive is to analyse one such random property of data transfer in case of search queries typed in a search box.
- This provides us a means of security analysis of a search engine.



# Side Channel Attacks

## Introduction

Side Channel Attack (SCA) is basically exploiting the information hidden in the physical implementation of a system instead of the algorithmic implementation.

In this project the data packets exchanged between the *server* and *client* serve as a side channel for analysis.



# Approach

# O1

## Analyzing Data Packets using Wireshark

- Analysed *Application Data Packets* based on source, destination and length fields.
- Observed a pattern in the length of data packets exchanged for a specific query.

10	0.318102047	192.168.43.185	204.79.197.200	TCP	56 56496 → 443 [ACK]	Seq=158 Ack=736 Win=2040 Len=0
13	2.108403775	192.168.43.185	204.79.197.200	TLSv1.2	214 Application Data	
14	2.193985647	204.79.197.200	192.168.43.185	TCP	56 443 → 56496 [ACK]	Seq=736 Ack=316 Win=1025 Len=0
15	2.567316178	204.79.197.200	192.168.43.185	TLSv1.2	751 Application Data	
16	2.567376771	192.168.43.185	204.79.197.200	TCP	56 56496 → 443 [ACK]	Seq=316 Ack=1431 Win=2061 Len=0
17	2.567413684	204.79.197.200	192.168.43.185	TLSv1.2	94 Application Data	
18	2.567426610	192.168.43.185	204.79.197.200	TCP	56 56496 → 443 [ACK]	Seq=316 Ack=1469 Win=2061 Len=0
23	5.126137928	192.168.43.185	204.79.197.200	TLSv1.2	214 Application Data	
26	5.200051400	204.79.197.200	192.168.43.185	TCP	56 443 → 56496 [ACK]	Seq=1469 Ack=474 Win=1024 Len=0
28	5.426278182	204.79.197.200	192.168.43.185	TLSv1.2	699 Application Data	
29	5.426320886	192.168.43.185	204.79.197.200	TCP	56 56496 → 443 [ACK]	Seq=474 Ack=2112 Win=2082 Len=0
30	5.426374828	204.79.197.200	192.168.43.185	TLSv1.2	94 Application Data	
31	5.426391611	192.168.43.185	204.79.197.200	TCP	56 56496 → 443 [ACK]	Seq=474 Ack=2150 Win=2082 Len=0
33	7.947608216	192.168.43.185	204.79.197.200	TLSv1.2	215 Application Data	
34	8.015466769	204.79.197.200	192.168.43.185	TCP	56 443 → 56496 [ACK]	Seq=2150 Ack=633 Win=1024 Len=0
35	8.387310410	204.79.197.200	192.168.43.185	TLSv1.2	719 Application Data	
36	8.387328583	192.168.43.185	204.79.197.200	TCP	56 56496 → 443 [ACK]	Seq=633 Ack=2813 Win=2103 Len=0
37	8.387339163	204.79.197.200	192.168.43.185	TLSv1.2	94 Application Data	
38	8.387341663	192.168.43.185	204.79.197.200	TCP	56 56496 → 443 [ACK]	Seq=633 Ack=2851 Win=2103 Len=0

▶ Frame 15: 751 bytes on wire (6008 bits), 751 bytes captured (6008 bits) on interface 0  
▶ Linux cooked capture  
▶ Internet Protocol Version 4, Src: 204.79.197.200, Dst: 192.168.43.185  
▶ Transmission Control Protocol, Src Port: 443, Dst Port: 56496, Seq: 736, Ack: 316, Len: 695  
▼ Secure Sockets Layer  
    ▼ TLSv1.2 Record Layer: Application Data Protocol: http-over-tls  
        Content Type: Application Data (23)  
        Version: TLS 1.2 (0x0303)  
        Length: 690  
        Encrypted Application Data: 000000000000008de894b598a25ad0dec19c71fb630d8621...

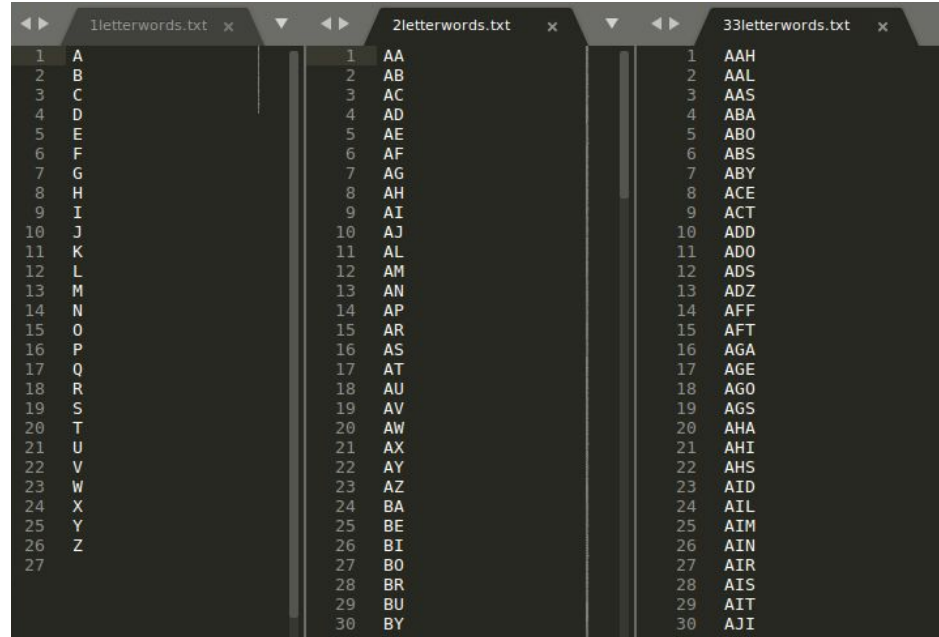
0030	50 18 04 01 4a 58 00 00 17 03 03 02 02 00 00 00	P...JX...
0040	00 00 00 00 8d e8 94 b5 98 a2 5a d0 de c1 9c 71	.....Z....q
0050	fb 63 0d 86 21 e2 e3 64 30 d7 7a b1 6b 39 f4 8f	.c...!..d 0 z.k9..

Fig1. A Wireshark window showing live capture.

## 02

# Generating Test Strings

- Extracted valid english words of length maximum to three from dictionary .
- Generated one letter prefixes for valid two letter words.
- Generated two letter prefixes for valid three lettered words.



1letterwords.txt	2letterwords.txt	33letterwords.txt
1 A	1 AA	1 AAH
2 B	2 AB	2 AAL
3 C	3 AC	3 AAS
4 D	4 AD	4 ABA
5 E	5 AE	5 ABO
6 F	6 AF	6 ABS
7 G	7 AG	7 ABY
8 H	8 AH	8 ACE
9 I	9 AI	9 ACT
10 J	10 AJ	10 ADD
11 K	11 AL	11 ADO
12 L	12 AM	12 ADS
13 M	13 AN	13 ADZ
14 N	14 AP	14 AFF
15 O	15 AR	15 AFT
16 P	16 AS	16 AGA
17 Q	17 AT	17 AGE
18 R	18 AU	18 AGO
19 S	19 AV	19 AGS
20 T	20 AW	20 AHA
21 U	21 AX	21 AHI
22 V	22 AY	22 AHS
23 W	23 AZ	23 AID
24 X	24 BA	24 AIL
25 Y	25 BE	25 AIM
26 Z	26 BI	26 AIN
	27 BO	27 AIR
	28 BR	28 AIS
	29 BU	29 AIT
	30 BY	30 AJI

Fig2. Text files containing test strings.





# 03

## Making a Character - Data Packet Length map

- Each character is responsible for generating some Application Data Packets whose lengths lie under a observable range.
- So a mean of the values can be mapped to each unique character.

Test Characters	Application Packet Data Lengths					
	Trial #1	Trial #2	Trial #3	Trial #4	Trial #5	
A	475	664	682	668	662	
B	639	645	684	643	644	
C	657	660	656	655	655	
D	648	645	649	650	645	
E	634	630	629	633	628	
F	652	650	652	649	651	
G	648	644	644	649	650	
H	651	646	648	646	651	
I	656	652	654	651	650	
J	634	636	630	630	630	
K	661	660	657	660	658	
L	708	708	706	707	704	
M	856	630	623	625	623	
N	623	625	619	619	620	
O	619	615	621	617	615	
P	658	653	653	656	652	
Q	642	645	641	647	641	
R	624	851	620	617	617	
S	691	692	691	686	688	
T	649	650	557,128	649	649	
U	665	666	663	666	661	



## Next Steps



04

## Generating Data Set

- Manual checking showed that each character can be mapped to an observable range of data packet lengths.
- Same could be extended to multi-character strings.

# STEPWISE IMPLEMENTATION :

## 4.1 Automatically feeding Test Strings to the website search box.


- A Javascript snippet will iterate through files containing Test Strings and feed them to the website's search box letter by letter (at some specified interval).

## 4.2 Catching the concerned packets for each test string.

- A background PyShark snippet will capture the newly arrived packets as a character is fed to the search box.

## 4.2 Mapping strings with their packet length range.

- Making a map that stores all the test strings with their average data packet length. This is the proposed DATA-SET.



## 05 Finding the probable number of letters based on the sequence of packet lengths.

- For one letter typed we capture the live packet length. Propose a tolerance parameter (eg +5 and -5) and check the characters whose average packet length lie in this range.
- Select the top 10 characters in the closest proximity of observed length to generate a set.
- The same can be done for two and three letter words with the condition that the prefix should be present in the previously generated set.

# Conclusion

- After generating the set we select top 5 most probable words as depicted by our approach and check whether it matches the entered query.
- Analyse the percentage of cases in which this approach is correctly predicting the query. If the percentage is high that means the search box is susceptible to data leak.



# Limitations

- These should not be any spaces or backspaces in the typed query.
  - Query typed is maximum of length three.
  - The query must contain only valid english words.
  - The wait time between typing of two characters depends on server response time.
-



# References

- <http://nsw.scrabble.org.au/backup/2&3&4%20complimentary.pdf>
- <https://www.lifewire.com/wireshark-tutorial-4143298>
- <https://thepacketgeek.com/series/intro-to-pyshark/>
- [https://www.javatpoint.com/document-getElementById\(\)-method](https://www.javatpoint.com/document-getElementById()-method)
- <https://drive.google.com/open?id=0B3vCfreSb--RZFVQd0UzM3VITjk3MkkxLTA0enR1Vk0yTIVz>





# Thank You !