



INDIAN INSTITUTE OF TECHNOLOGY, ROORKEE
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Report on Project Assignment

CSN-341 : Computer Networks

Name : Shubhang Tripathi
Branch : Computer Science and Engineering
Enrollment Number : 18114074
Batch : CS-2
Email : stripathil@cs.iitr.ac.in
Phone Number : +91-7456099390

Under the guidance of:
Dr. Rajdeep Niyogi

Contents

1	System Configuration	1
2	Problem Statement 1	1
2.1	Problem Description	1
2.2	HTTP	1
2.2.1	Experiment Procedure	1
2.2.2	Results and Inference	3
2.3	HTTPs	3
2.3.1	Experiment Procedure	3
2.3.2	Results and Inference	4
2.4	TELNET	5
2.4.1	Experiment Procedure	5
2.4.2	Results and Inference	6
2.5	FTP	6
2.5.1	Experiment Procedure	6
2.5.2	Results and Inference	7
2.6	SSH	8
2.6.1	Experiment Procedure	8
2.6.2	Results and Inference	9
3	Problem Statement 2	10
3.1	Problem Description	10
3.2	Experiment Procedure	10
3.3	Results	11
3.4	Inferences Drawn	11
3.5	Guidelines to run the code	12
4	Conclusion	13

1 System Configuration

The complete configuration info of my system is as follows:

Table 1: System Configuration

Operating System	Ubuntu 20.04.1 LTS
Kernel	GNU/Linux 5.4.0-53-generic
Host	HP Pavilion Notebook
CPU	Intel i7-6500U (4) @ 3.100GHz
Memory	7801MiB
Architecture	x86_64
CPU op-mode(s)	32-bit, 64-bit
Byte Order	Little Endian
Network Controller	Broadcom Inc. and subsidiaries BCM43142 802.11b/g/n (rev 01)
Ethernet Controller	RTL810xE PCI Express Fast Ethernet controller (rev 0a)
golang Compiler Version	go1.14.1 linux/amd64
Wireshark Version	Wireshark 3.2.7 (Git v3.2.7)

2 Problem Statement 1

2.1 Problem Description

To demonstrate password sniffing over different application layer protocols like HTTP, HTTPs, TELNET, FTP, SSH and showing if user sensitive data(like username and passwords) can be captured by anyone monitoring the traffic or not in each of them.

2.2 HTTP

Hypertext **T**ransfer **P**rotocol (HTTP) is an application layer protocol. It is used to send and receive webpages and files on the internet. It was developed by Tim Berners-Lee and is now coordinated by the W3C. HTTP version 1.1 is the most common used version today. It is defined in RFC 2616. This protocol by itself does not support hiding of any user sensitive data. So, it should be possible to obtain such information for a person monitoring the traffic of a network.

2.2.1 Experiment Procedure

For this procedure, I have taken <http://2019shell1.picoctf.com:12273/login.html> as an example of a HTTP login page.

- First we start capturing packets via wireshark on the **wlo1** interface.
- Next, we visit the sample [page](#). We enter the credential "admin" and "admin", and click on the "Login" button¹.
- Next, we look at the source code² of the page and find that it uses the HTTP **POST** method to send the login data.

- Next, we stop capturing the packets and save it to a pcapng file, so that I can analyze it further in wireshark. In my case the file is saved as "http.pcapng".
- Next, we open the pcapng file in wireshark. Since we know that the HTTP post method was used, we can use the filter "http.request.method == POST". Now, there is only one request. Using wireshark, we can analyze the HTML form url part as shown in the figure. We can clearly see³ the credentials we entered i.e. "username=admin" and "password=admin".

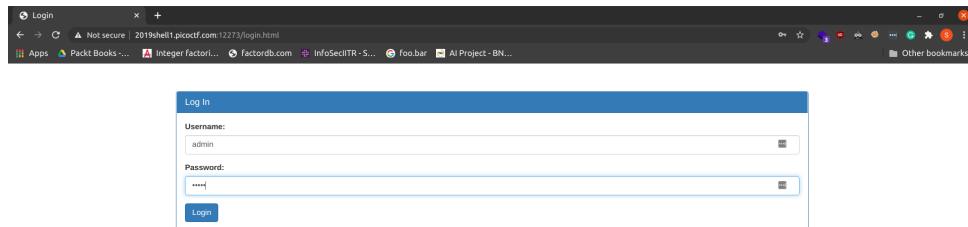


Figure 1: HTTP Login Page

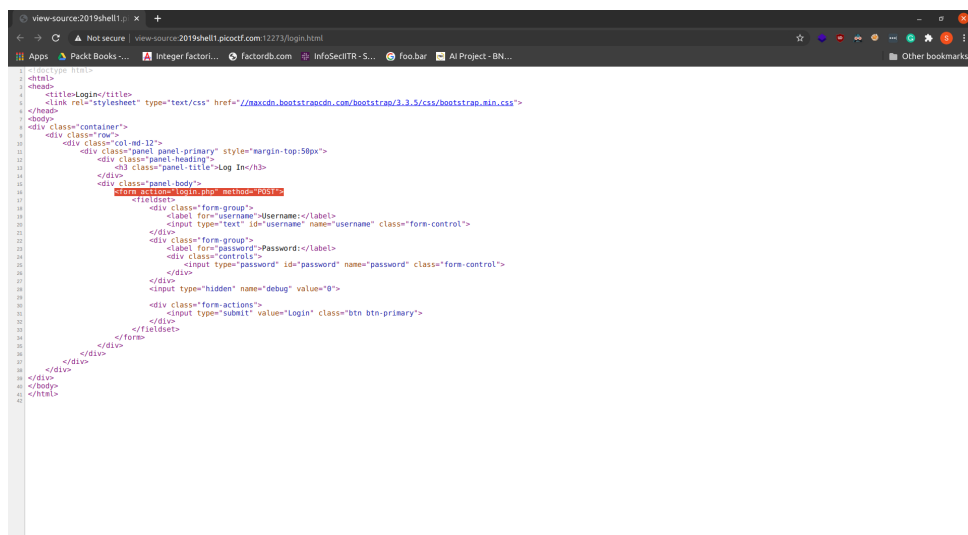


Figure 2: HTTP Page Source

2.2.2 Results and Inference

Thus, we see that the user sensitive data can be easily captured by anyone monitoring the data as in the screenshot below³.

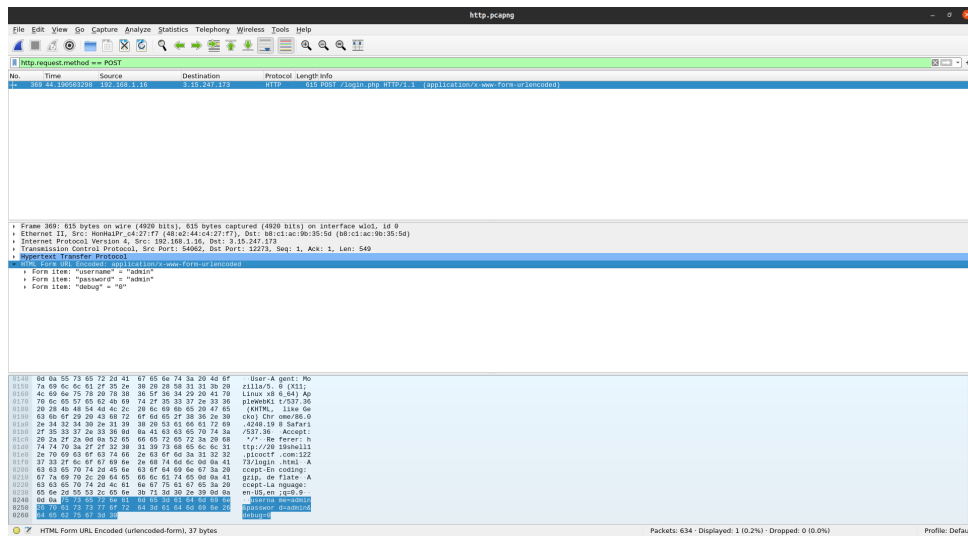


Figure 3: Wireshark Analysis of HTTP

2.3 HTTPs

Hypertext Transfer Protocol secure (HTTPS) is the secure version of HTTP, which is the primary protocol used to send data between a web browser and a website. HTTPS is encrypted in order to increase security of data transfer. This is particularly important when users transmit sensitive data, such as by logging into a bank account, email service etc. Thus, in theory, we should not be able to get any user credentials by sniffing the packets being sent.

2.3.1 Experiment Procedure

For this procedure, I have taken <https://2019game.picocft.com/> as an example of a HTTPs login page.

- First we start capturing packets via wireshark on the **wlo1** interface.
- Next, we visit the sample [page](#). We enter the credential "Th3F0x" and my password (not disclosed here) , and click on the "Login" button⁴.
- Next, we stop capturing the packets and save it to a pcapng file, so that I can analyze it further in wireshark. In my case the file is saved as "https.pcapng".
- Here, we can select any packet -> right click it -> follow -> TCP Stream. We can now see the complete TCP conversation that took place. We see⁵ that it is not recognizable at all, i.e. everything looks like random values. This is due to the TLS encryption.

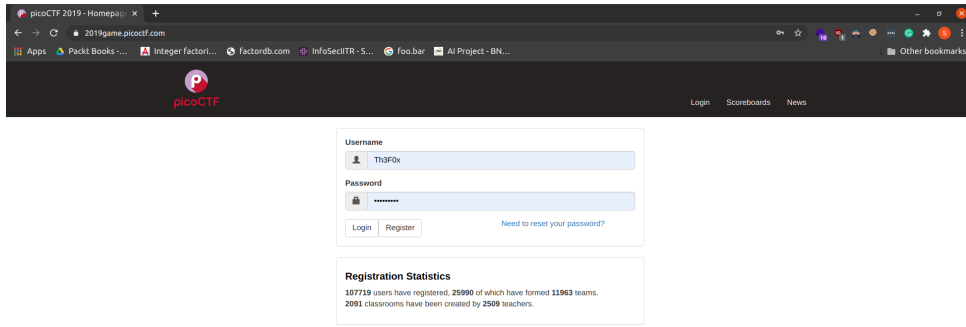


Figure 4: HTTPs Login Page

2.3.2 Results and Inference

Thus we see that not only sensitive credentials, no data related to the HTTP conversation can be viewed by someone who is monitoring the network. This is shown in the figure below⁵. Here the red colored text represent data sent by us.

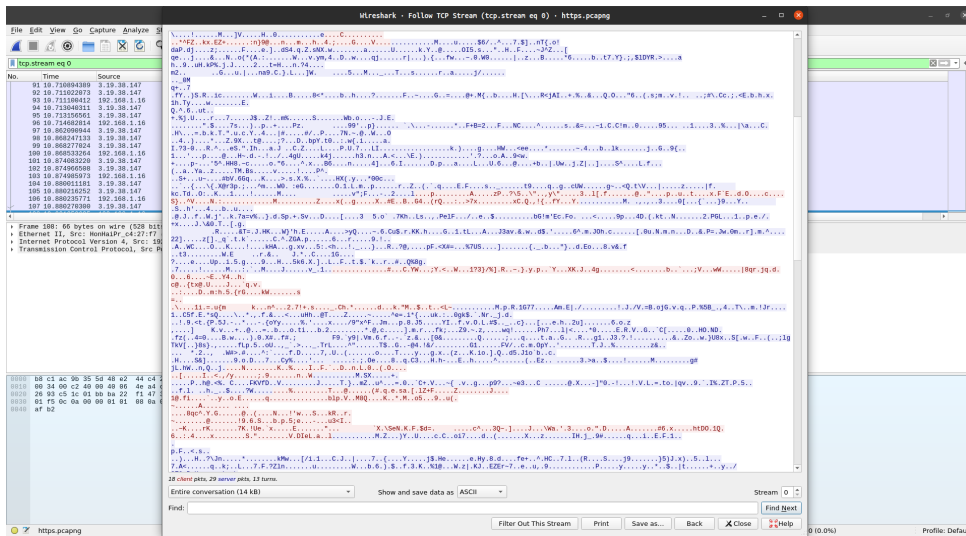


Figure 5: Wireshark Analysis of HTTPs

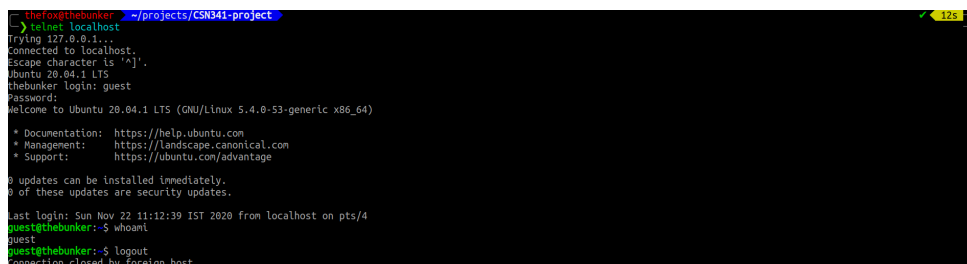
2.4 TELNET

Teletype Network (TELNET) is one of the earliest remote login protocols on the Internet. It was initially released in the early days of IP networking in 1969, and was for a long time the default way to access remote networked computers. It is a client-server protocol that provides the user a terminal session to the remote host from the telnet client application. Like HTTP, this protocol also offers no built-in features to encrypt/hide user data in any way. Thus, we should be able to get user credentials while sniffing network packets.

2.4.1 Experiment Procedure

For this procedure, I have used a TELNET server hosted on my own machine as an example. To do this, I had to use the command "*sudo apt install telnetd*" on my machine. I have also used a guest user on my machine for this example.

- First we start capturing packets via wireshark on the **loopback** interface. Unlike the previous two, we don't use the "wlo1" interface as this TELNET server is not hosted in the internet, rather my own machine, which the loopback interface corresponds to.
- Next, use the "*telnet*" command to establish a TELNET connection to the server. We use it like "telnet localhost" as shown in the figure⁶. We enter the credential "guest" and "guest".
- We are successfully authenticated as shown in the figure⁶, and then we close the connection.
- Next, we stop capturing the packets and save it to a pcapng file, so that I can analyze it further in wireshark. In my case the file is saved as "telnet.pcapng".
- Here, we can select any packet -> right click it -> follow -> TCP Stream. We can now see the complete TCP conversation that took place. We see⁷ that our password is clearly visible as '*guest*', our username appears as '*ggwueesstt*'. Thus the characters in our username seem to be viewed as twice. This is because whatever username we write is also echoed back to our screen as in figure⁶. Thus one can easily figure out the username to be '*guest*'.



```
therfox@thebunker: ~/projects/CSN341-project
-> telnet localhost
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
thebunker login: guest
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-53-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

0 updates can be installed immediately.
0 of these updates are security updates.

Last login: Sun Nov 22 11:12:39 IST 2020 from localhost on pts/4
guest@thebunker:~$ whoami
guest
guest@thebunker:~$ logout
Connection closed by foreign host.
```

Figure 6: TELNET Connection

2.4.2 Results and Inference

Thus we see that the user sensitive credentials along with commands and their output can easily be viewed by someone who is monitoring the network. This is shown in the figure below⁷. Here the red colored text represent data sent by us, and the blue colored text is the data sent by the server.



Figure 7: Wireshark Analysis of TELNET

2.5 FTP

File Transfer Protocol (FTP) is a standard Internet protocol for transmitting files between computers on the Internet over TCP/IP connections. FTP is a client-server protocol where a client will ask for a file, and a local or remote server will provide it. Just like HTTP and TELNET, this protocol is also insecure in terms of hiding user details. So, we should be able to sniff user credentials from the packets.

2.5.1 Experiment Procedure

For this procedure, I have used an FTP server hosted on my own machine as an example. To do this, I had to use the command "*sudo apt install vsftpd*" on my machine. I have also used a guest user on my machine for this example.

- First we start capturing packets via wireshark on the **loopback** interface. Like the previous case, we don't use the "wlo1" interface as this FTP server is not hosted in the internet, rather my own machine, which the loopback interface corresponds to.
- Next, use the "*ftp*" command to establish an FTP connection to the server. We use it like "*ftp localhost*" as shown in the figure⁸. We enter the credential "guest" and "guest".
- We are successfully authenticated as shown in the figure⁸, and then we close the connection.
- Next, we stop capturing the packets and save it to a pcapng file, so that I can analyze it further in wireshark. In my case the file is saved as "*ftp.pcapng*".

- Here, we can select any packet -> right click it -> follow -> TCP Stream. We can now see the complete TCP conversation that took place. We see⁹ that our password is clearly visible as 'guest', our username as 'guest'. Thus our credentials are very clearly visible in the captured packets.

```

therox@thebunker: ~/projects/CSN341-project
> ftp localhost
Connected to localhost.
220 (vsFTPd 3.0.3)
Name (localhost:therox): guest
131 Please specify the password.
Password:
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> ls
200 PORT command successful. Consider using PASV.
150 Here comes the directory listing.
-rw-r--r-- 1 1801 1801 8980 Nov 22 11:12 examples.desktop
226 Directory send OK.
ftp> exit
221 Goodbye.

```

Figure 8: FTP Connection

2.5.2 Results and Inference

Thus we see that the user sensitive credentials can easily be viewed by someone who is monitoring the network. This is shown in the figure below⁹. Here the red colored text represent data sent by us, and the blue colored text is the data sent by the server.

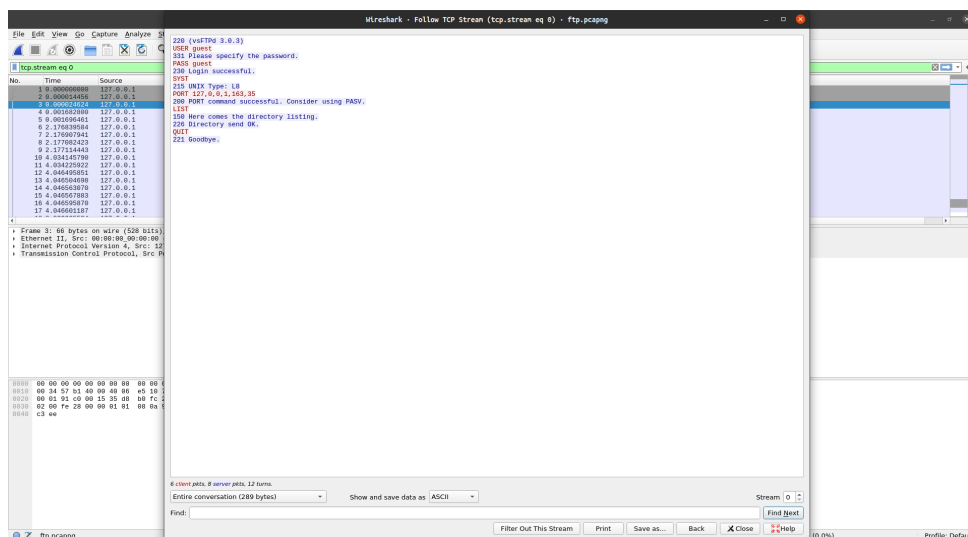


Figure 9: Wireshark Analysis of FTP

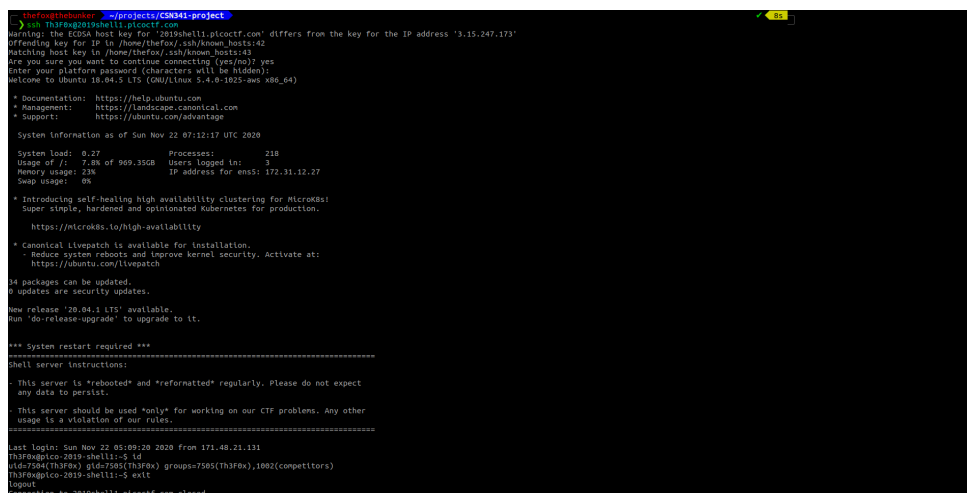
2.6 SSH

The **Secure Shell** (SSH) protocol uses encryption to secure the connection between a client and a server. All user authentication, commands, output, and file transfers are encrypted to protect against attacks in the network. Thus, in theory, we should not be able to get user sensitive data from the packets

2.6.1 Experiment Procedure

For this procedure, I have taken the domain `ssh://2019shell1.picoc.tf.com` as an example of an SSH server.

- First we start capturing packets via Wireshark on the **wlo1** interface. This is because the server is hosted on the internet and I am connected to the internet via WiFi.
- Next, use the `ssh` command to establish an ssh connection to the server. My username on the server is `Th3F0x`. We connect via ssh like `ssh Th3F0x@2019shell1.picoc.tf.com` as shown in the figure¹⁰. We enter our password at the prompt (password not disclosed here).
- We are successfully authenticated as shown in the figure¹⁰, and then we close the connection.
- Next, we stop capturing the packets and save it to a pcapng file, so that I can analyze it further in Wireshark. In my case the file is saved as `ssh.pcapng`.
- Here, we can select any packet -> right click it -> follow -> TCP Stream. We can now see the complete TCP conversation that took place. We see¹¹ that just like *HTTPs*, the data here also looks like random values, nothing like the input we sent or the output we received. This is because unlike *TELNET*, **SSH** applies a layer of encryption on our packets.



```
th3f0x@ubuntu: ~/projects/CN341-project
$ ssh Th3F0x@2019shell1.picoc.tf.com
Warning: the ECDSA host key for '2019shell1.picoc.tf.com' differs from the key for the IP address '3.15.247.173'
Offering key for IP in /home/th3f0x/.ssh/known_hosts:42
Matching host key in /home/th3f0x/.ssh/known_hosts:42
Are you sure you want to continue connecting (yes/no)? yes
Enter your platform password (characters will be hidden):
Welcome to Ubuntu 18.04.5 LTS (GNU/Linux 5.4.0-1025-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

System information as of Sun Nov 22 07:12:17 UTC 2020

System load: 0.27          Processes: 218
Usage of /:  7.8% of 969.3GB  Users logged in: 3
Memory usage: 23%          IP address for ens3: 172.31.12.27
Swap usage:  0%

 * Introducing self-healing high availability clustering for MicroK8s!
   Super simple, hardened and opinionated Kubernetes for production.
   https://microk8s.io/high-availability

 * Canonical Livepatch is available for installation.
   - Reduce system reboots and improve kernel security. Activate at:
     https://ubuntu.com/livepatch

14 packages can be updated.
0 updates are security updates.

New release '20.04.1 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

*** System restart required ***

=====
Shell server instructions:

 * This server is "rebooted" and "reformatted" regularly. Please do not expect
   any data to persist.

 * This server should be used "only" for working on our CTF problems. Any other
   usage is a violation of our rules.
=====

Last login: Sun Nov 22 05:09:20 2020 from 171.48.21.131
th3f0x@2019-shell1-5:~$ id
uid=7504(th3f0x) gid=7505(th3f0x) groups=7505(th3f0x),1002(competitors)
th3f0x@2019-shell1-5:~$ exit
logout
Connection to 2019shell1.picoc.tf.com closed.
```

Figure 10: SSH Login

2.6.2 Results and Inference

Thus we see that the user sensitive credentials can not be viewed by someone who is monitoring the network due to the encryption that takes place. This is shown in the figure below¹¹. Here the red colored text represent data sent by us, and the blue colored text is the data sent by the server.

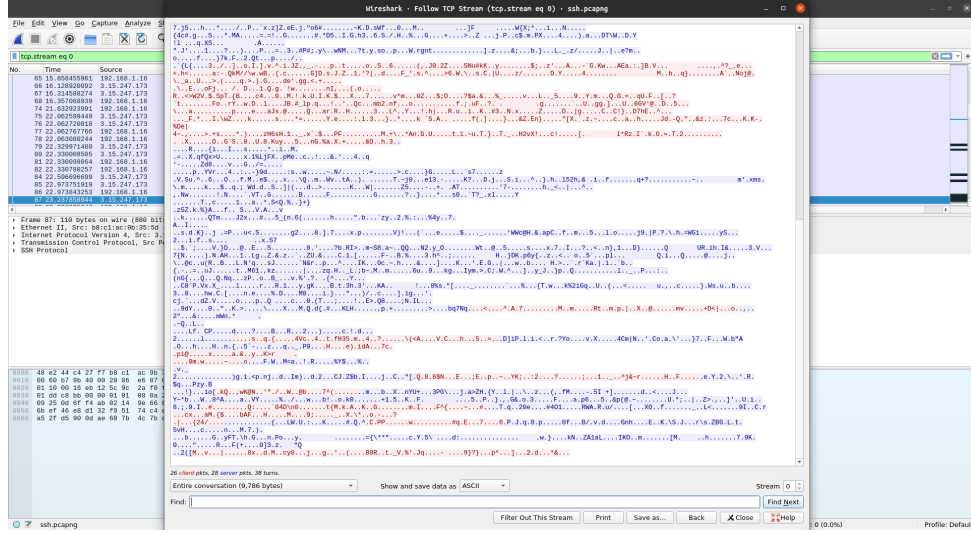


Figure 11: Wireshark Analysis of SSH

3 Problem Statement 2

3.1 Problem Description

An organization wants to keep track that its employee's access what all websites in there day to day life. Your task is to identify the top 3 websites being visited by the employees collectively as a whole. This can be simulated using a packet sniffer tool such as Wireshark. Try to capture packet in your system for about 4-5 hours each and collectively based on the IPs and MAC addresses of the systems, analyze the top 3 regularly visited websites.

3.2 Experiment Procedure

Monitor mode, or RFMON (Radio Frequency MONitor) mode, allows a computer with a wireless network interface controller (WNIC) to monitor all traffic received on a wireless channel. If a wifi-adapter supports this mode, it can be used to monitor all traffic on a wireless network. In my case, I have captured packets from my own system, as my wifi-adapter does not support this mode of operation.

The steps involved in doing the experiment for the given problem statement are:

- Firstly, I needed to capture some packets from my system. To do this, I started capturing packets via wireshark on the **wlo1** interface i.e. the Wi-Fi interface as I was connected to my home Wi-Fi network. After that I started going on about my regular work, while the packet capture process was going on in the background.
- Then, after around 4 hours, I stopped the packet capture process. Then in order to save these packets for later use, I saved them by clicking on the "save this packet capture" located in the menu toolbar, I saved it in the default *pcapng* format, so that I could look at the packets again at a later time via wireshark itself. In my case, I saved it as "*capture.pcapng*".
- Next, I opened the saved "capture.pcapng" file in wireshark. My task was to find most visited websites. In case of http i can do this by looking for the *hostname* http header. But, in todat's scenario, most websites use *https*, in which case the raw http headers are not visible due to encryption. So, I just filtered out the ssl packets by applying the filter *tls*. Now, all packets were most probably corresponding to visiting a website. (Note: This is just a measure taken due to **tls** complications, may not correspond to correct count).
- Next, I exported these packets in a form which was easy to process via programming. I could have chosen *csv*, *xml* or *json*. I my case, I chose JSON and exported it by going to "File" -> "Export Packet Dissections" -> "As JSON" and exported all these packets in JSON format into a file. In my case the file is "capture.json"
- Now, all that was left was to write some code to find out the count of visits to different IP addresses, then find out top 3 counts and their IPs. After that, the code would try to do a reverse IP lookup to try to find a domain name. For this code, i decided to use the [go](#) programming language. The main reason for using this language was that it is a compiled language, so it would run faster compared to something like python and also that I can compile my code for any Operating System, and also for any architecture, without need of special toolchains for each of

them separately.

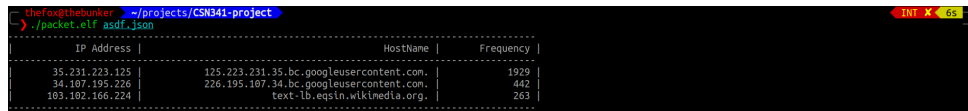
3.3 Results

After running the code, I got the following results:

Table 2: Results of finding the top-3 websites in packet capture

IP Address	HostName	Frequency
35.231.223.125	125.223.231.35.bc.googleusercontent.com	1929
34.107.195.226	226.195.107.34.bc.googleusercontent.com	442
103.102.166.224	text-lb.eqsin.wikimedia.org	263

A screenshot of the results is also given below:



IP Address	HostName	Frequency
35.231.223.125	125.223.231.35.bc.googleusercontent.com	1929
34.107.195.226	226.195.107.34.bc.googleusercontent.com	442
103.102.166.224	text-lb.eqsin.wikimedia.org	263

Figure 12: Code Output

3.4 Inferences Drawn

- We see that none of these hostnames match with anything normal website we usually hear about. This is because most websites point their domain names to different hostnames for different regions. We can google these hostnames, to try to find out which ones these correspond to. For e.g., if we google "125.223.231.35.bc.googleusercontent.com", we can find that it corresponds to <https://overleaf.com>. A screenshot of the same is attached below. This is accurate, as I remember using overleaf that day to write a report. Thus using these hostnames, we can find more accurate versions of what websites may have been visited.

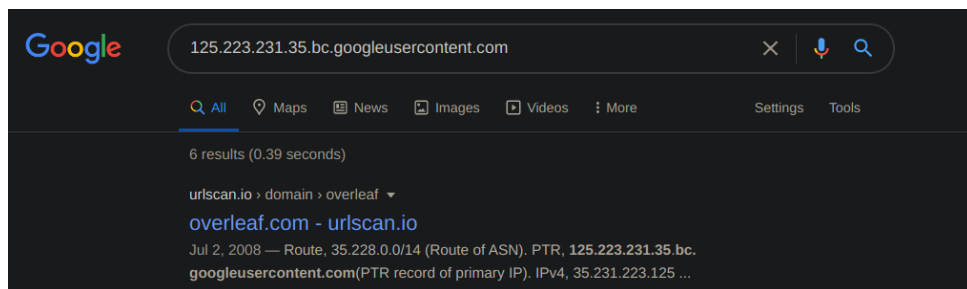


Figure 13: Hostname Reconnaissance

- We see that since most websites use https today, we can't just see the HTTP header for hostname, as the HTTP layer is transported in encrypted form.
- Even though we weren't able to view the Hostname header, we could still use the IP information available in the packets to find out about the websites being visited.

3.5 Guidelines to run the code

The code written needs a json file to parse. In order to convert the given pcapng to required json format, we need to first apply the tls filter as in the figure below.

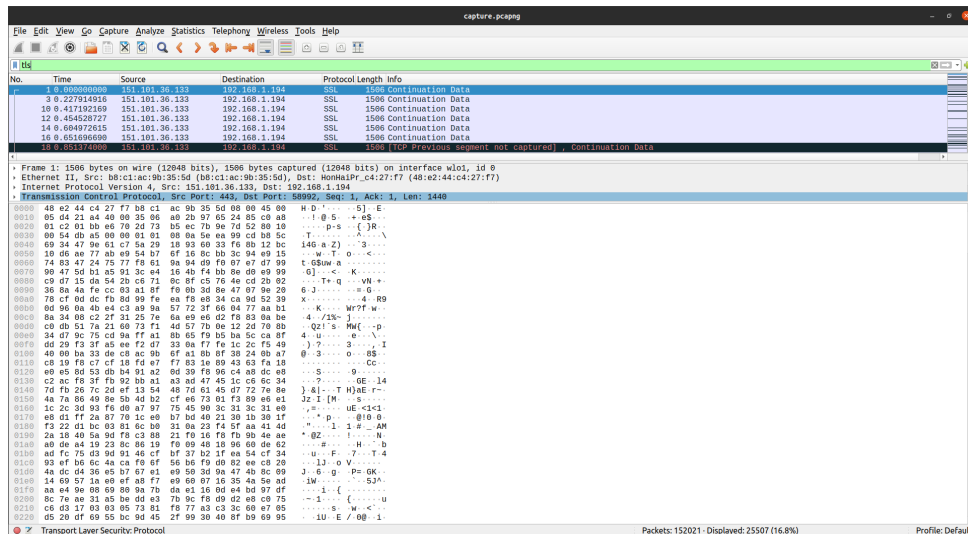


Figure 14: Applying tls filter

Then we need to go to "File" -> "Export Packet Dissections" -> "As JSON" to get a JSON output of required packets.

The code was written in the [go](#) programming language. In order to compile this, following steps need to be followed:

- Firstly, install the *go* compiler for your operating system/architecture from [this](#) website.
- Make sure the *go* program is on your system path. You can check this by typing "go version" in your batch/powershell/linux/OSX terminal. It should print something like "go version go1.<version>.1 <OS>/<ARCH>". If it doesn't consult google on how to add go to path.
- After go has been added to the path, we can just go to the folder containing the "packet.go" file and run "go build packet.go" to compile the binary for your machine. Once compiled you can run it via ".\package <json file name>" on windows or "./package <json file name>" on linux/macOS.

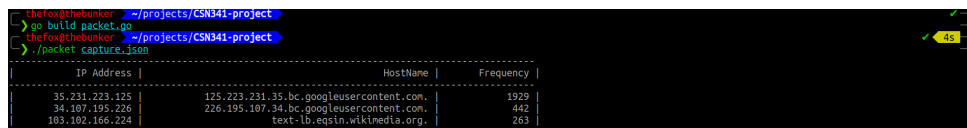


Figure 15: Building and Running The Code

4 Conclusion

In this report, I had two problem statements.

- From the first problem statement, I learned about the various application layer protocols, their structure, about their client and server utilities. Most importantly, I learned about the various old protocols like HTTP, TELNET, FTP etc which were not secure i.e. user credentials could be easily mined by someone listening to the network. However, recently, we have switched to more secure protocols which provide the same functionality like HTTPS in place of HTTP and SSH in place of TELNET, which also provide security against such attacks.
- From the second problem statement, we see that today, since most websites use HTTPS instead of HTTP, it is not easy to find the websites being visited. In earlier days of plain HTTP, one could just look at the Hostname header to find this information, but this is not possible in HTTPS. So, what we did was, we took all the TLS packets and analysed the IP header information, to find the frequencies of all the IPs visited. Then we reverse looked up these IPs to find the website names. This did not end up giving accurate info straight away, but on further research we were able to further decode these hostnames. So, although it may be difficult, it is still possible to find out the websites being visited.

In the end, I would like to thank Dr. Rajdeep Niyogi for this learning opportunity to work on this project to better understand the Networking concepts and for his support.

Bibliography

- [1] *HTTP* Hypertext Transfer Protocol
https://simple.wikipedia.org/wiki/Hypertext_Transfer_Protocol
- [2] *HTTPS* What Is HTTPS?
<https://www.cloudflare.com/learning/ssl/what-is-https/>
- [3] *TELNET* <https://www.ssh.com/ssh/telnet>
<https://www.ssh.com/ssh/telnet>
- [4] *FTP* What is FTP (File Transfer Protocol)
<https://searchnetworking.techtarget.com/definition/File-Transfer-Protocol-FTP>
- [5] *SSH* Start page for the SSH (Secure Shell) protocol
<https://www.ssh.com/ssh/>
- [6] *Monitor Mode*
https://en.wikipedia.org/wiki/Monitor_mode