



Wireshark

A Pentester Guide

Table of Contents

Abstract.....	3
Introduction.....	4
Network Packet Forensic.....	4
Examine Layers captured by Wireshark.....	5
Ethernet Header (Data Link)	6
IP Header (Network Layer).....	7
TCP Header (Transport Layer)	9
Structure of TCP segment.....	9
Different Types of TCP flags	10
Password Sniffing	12
Capture HTTP Password	12
Dissect HTTPS Packet Captures	14
Capture Telnet Password	18
Capture FTP Password	19
Capture SMTP password.....	20
Analyzing SNMP Community String.....	23
Capture MSSQL Password	25
Capture PostgreSQL Password	26
Creating Firewall Rules with Wireshark	28
Conclusion	29
References	29

Abstract

Many people wonder if Wireshark can capture passwords. The answer is undoubtedly yes! Wireshark can capture not only passwords, but any type of data passing through a network – usernames, email addresses, personal information, pictures, videos, or anything else.

Wireshark can sniff the passwords passing through as long as we can capture network traffic. But the question is, what kind of passwords are they? Or, more precisely, which network protocols' passwords can we obtain? That is the subject of this report.

Disclaimer: This report is provided for educational and informational purpose only (Penetration Testing). Penetration Testing refers to legal intrusion tests that aim to identify vulnerabilities and improve cybersecurity, rather than for malicious purposes.

Introduction

In the first section of this report, we'll delve into 'Network Packet Forensics,' exploring vital aspects such as data transfer between nodes, the 'OSI 7-layer model,' and how Wireshark stores information across layers when capturing network traffic.

Moving on to the second part, we'll examine how Wireshark can capture passwords, a result of certain unencrypted network protocols known as clear text protocols. These protocols expose all data, including passwords, making it visible to anyone with the ability to intercept the communication, such as a man-in-the-middle.

Network Packet Forensic

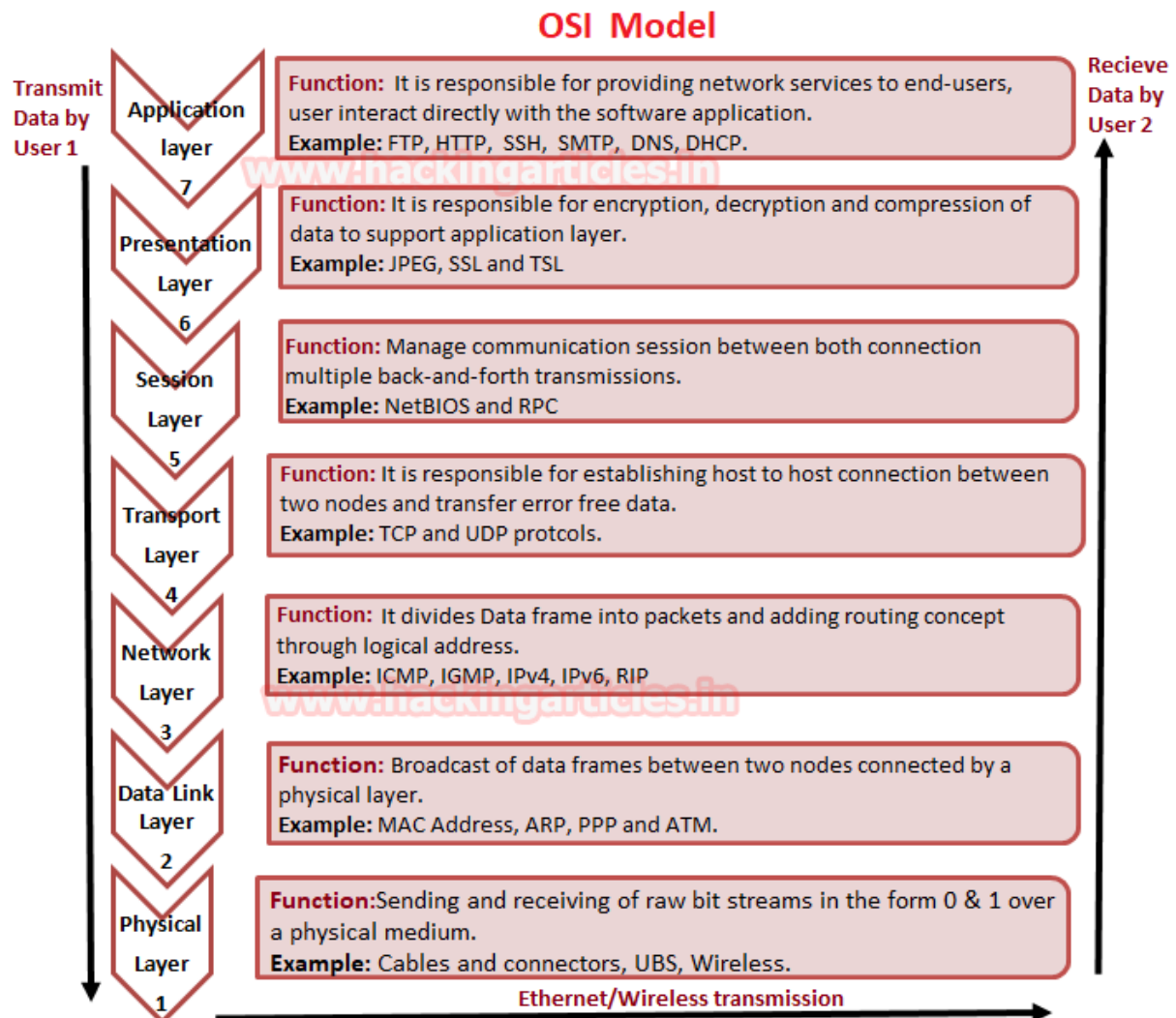
As we know for transferring the data from one system to other, we need a network connection which can be wired or wireless connection. But in the actual transmission of data does not only depend upon network connection apart from that it involves several phases for transmitting data from one system to another which was explained by the OSI model.

OSI stands for **Open Systems Interconnection** model which is a conceptual model that defines and standardizes the process of communication between the sender's and receiver's system. The data is transfer through 7 layers of architecture where each layer has a specific function in transmitting data over the next layer.

Now have a look over given below image where we had explained the functionality of each layer in the OSI model. So, when data is transmitted by sender's network then it will go in downward direction and data move from application layer to physical layer whereas when the receiver will receive the transmitted data it will come in an upward direction from physical layer to application layer.

Flow of Data from Sender's network: **Application > Presentation > Session > Transport > Network > Data Link > Physical**

Flow of Data from Receiver's network: **Physical > Data Link > Network > Transport > Session > Presentation > Application**



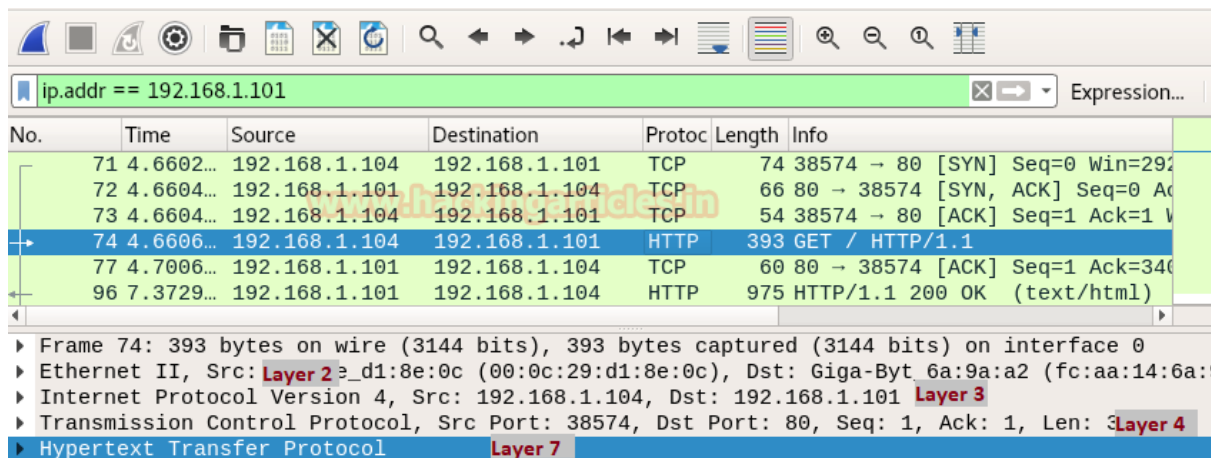
Examine Layers captured by Wireshark

Basically, when a user opens an application for sending or receiving Data then he directly interacts with the application layer for both operations either sending or receiving of data. For example, we act as a client when use Http protocol for uploading or Downloading a Game; FTP for downloading a File; SSH for accessing the shell of the remote system.

While connecting with any application for sharing data between server and client we make use of Wireshark for capturing the flow of network traffic stream to examine the OSI model theory through captured traffic.

From given below image you can observe that Wireshark has captured the traffic of four layers in direction of the source (sender) to destination (receiver) network.

Here it has successfully captured **Layer 2 > Layer 3 > Layer 4** and then **Layer 7** information.



Ethernet Header (Data Link)

Data link layer holds 6 bytes of **Mac address** of sender's system and receiver's system with 2 bytes of **Ether type** is used to indicate which protocol is encapsulated i.e. IPv4/IPv6 or ARP.

In Wireshark Ethernet II layer represent the information transmitted over the data link layer. From given below image you can observe that highlighted lower part of Wireshark is showing information in Hexadecimal format where the first row holds information of Ethernet headers details.

So here you can get the source and destination Mac address which also available in Ethernet Header.

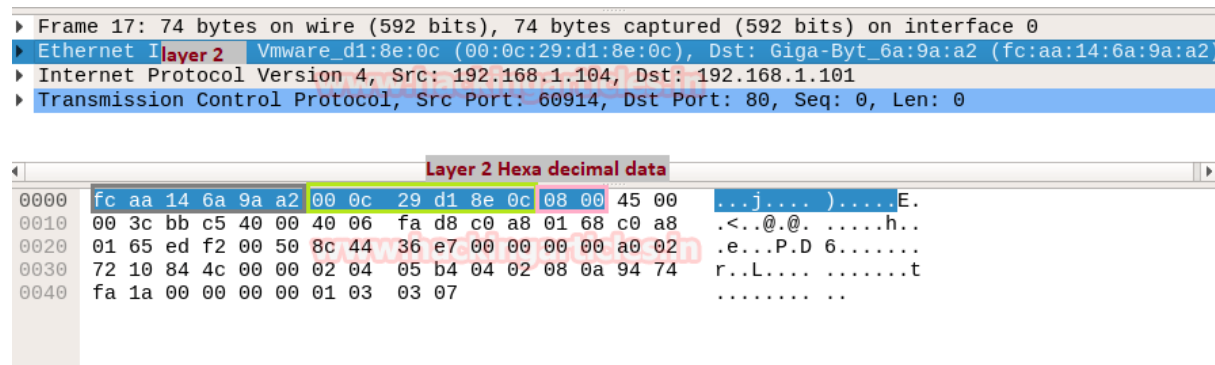
The row is divided into three columns as described below:

Ethernet header 14 bytes	Destination MAC Address 6 Bytes	Source MAC Address 6 Bytes	Ether Type 2 Bytes
Bits Color	Gray	Light Green	Pink
Hexadecimal value	Fc:aa:14:6a:9a:a2	00:0c:29:d1:8e:0c	0800

As we know the MAC address of the system is always represented in Hexadecimal format but both types are generally categorized in the ways given below:

Ether Type	Hexadecimal Value
ARP: Address Resolution Protocol	0x0806
IPv4: Internet Protocol version 4	0x0800
IPv6: Internet Protocol version 6	0x86dd
IEEE 802.1Q	0x8100

Once again if you notice the given below image then you can observe the highlighted text in pink colour is showing hex value **08 00** which indicates that here **IPv4** is used.



IP Header (Network Layer)

IP header in Wireshark has described the network layer information which is also known as the backbone of the OSI model as it holds Internet Protocol version 4's complete details. Network layer divides data frame into packets and defines its routing path through some hardware devices such as routers, bridges, and switches. These packets are identified through their logical address i.e. source or destination network IP address.

In the image of Wireshark, I have highlighted six most important values which contain vital information of a data packet and this information always flows in the same way as they are encapsulated in the same pattern for each IP header.

Now here, **45** represent IP header length where "4" indicates **IP version 4** and "5" is header length of **5 bits**. while **40** is time to live (**TTL**) of packet and **06** is hex value for **TCP** protocol which means these values changes if anything changes i.e. TTL, Ipv4 and Protocol.

Therefore, you can take help of given below table for examining TTL value for the different operating system.

Operating System	Hex Value TTL	Decimal value TTL
Windows	80	128
Linux	40	64
MAC	39	57

Similarly, you can take help of given below table for examining other Protocol value.

Protocol	Hex Value	Decimal Value
ICMP	1	1
TCP	6	6
EGP	8	8
UDP	11	17

From given below image you can observe Hexadecimal information of the IP header field and using a given table you can study this value to obtain their original value.

IP header (20 bytes)	Header length	Total Length	TTL	Protocol	Source IP	Destination IP
Bits Color	Red	Orange	Yellow	Dark Green	Dark Brown	Black
Hex Value	5	3c	40	06	C0.a8.01.68	C0.a8.01.65
Decimal value	5	60	64	6	192.168.1.104	192.168.1.105

The IP header length is always given in form of the bit and here it is 5 bytes which are also minimum IP header length and to make it 20 bytes, multiply 4 with 5 i.e., 20 bytes.

```

▶ Frame 17: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface 0
▶ Ethernet II, Src: Vmware_d1:8e:0c (00:0c:29:d1:8e:0c), Dst: Giga-Byt_6a:9a:a2 (fc:01:02:6a:9a:a2)
▶ Internet Protocol Version 4, Src: 192.168.1.104, Dst: 192.168.1.101 layer 3
▶ Transmission Control Protocol, Src Port: 60914, Dst Port: 80, Seq: 0, Len: 0

```

layer 3 Hexa decimal data															
0000	fc	aa	14	6a	9a	a2	00	0c	29	d1	8e	0c	08	00	45 00
0010	00	3c	bb	c5	40	00	40	06	fa	d8	c0	a8	01	68	c0 a8
0020	01	65	ed	f2	00	50	8c	44	36	e7	00	00	00	00	a0 02
0030	72	10	84	4c	00	00	02	04	05	b4	04	02	08	0a	94 74
0040	fa	1a	00	00	00	00	01	03	03	07					

TCP Header (Transport Layer)

Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) and Internet Control Message Protocol (ICMP) are the major protocols as it gives host-to-host connectivity at the Transport Layer of the OSI model. It is also known as Heart of OSI model as it plays a major role in transmitting errors free data.

By examining Network Layer information through Wireshark, we found that here TCP is used for establishing a connection with destination network.

We knew that a computer communicates with another device like a modem, printer, or network server; it needs to handshake with it to establish a connection.

TCP follows **Three-Way-Handshakes** as describe below:

- A client sends a TCP packet to the server with the **SYN flag**
- A server responds to the client request with the **SYN** and **ACK** flags set.
- Client completes the connection by sending a packet with the **ACK** flag set

Structure of TCP segment

Transmission Control Protocol accepts data from a data stream, splits it into chunks, and adds a TCP header creating a TCP segment. A TCP segment only carries the sequence number of the first byte in the segment.

A TCP segment consists of a segment header and a data section. The TCP header contains mandatory fields and an optional extension field.

Source Port	The 16-bit source port number, Identifies the sending port.
Destination Port	The 16-bit destination port number. Identifies the receiving port
Sequence Number	The sequence number of the first data byte in this segment. If the SYN control bit is set, the sequence number is the initial sequence number (n) and the first data byte is n+1.
Acknowledgment Number	If the ACK control bit is set, this field contains the value of the next sequence number that the receiver is expecting to receive.
Data Offset	The number of 32-bit words in the TCP header. It indicates where the data begins.
Reserved	Six bits reserved for future use; must be zero.

Flags	CWR, ECE, URG, ACK, PSH, RST, SYN, FIN
Window	Used in ACK segments. It specifies the number of data bytes, beginning with the one indicated in the acknowledgment number field that the receiver (the sender of this segment) is willing to accept.
Checksum	The 16-bit one's complement of the one's complement sum of all 16-bit words in a pseudo-header, the TCP header, and the TCP data. While computing the checksum, the checksum field itself is considered zero.
Urgent Pointer	Points to the first data octet following the urgent data. Only significant when the URG control bit is set.
Options	Just as in the case of IP datagram options, options can be either: <ul style="list-style-type: none"> – A single byte containing the option number – A variable length option in the following format
Padding	The TCP header padding is used to ensure that the TCP header ends and data begins on a 32-bit boundary. The padding is composed of zeros.

Different Types of TCP flags

TCP flags are used within TCP header as these are control bits that specify particular connection states or information about how a packet should be set. TCP flag field in a TCP segment will help us to understand the function and purpose of any packet in the connection.

List of flags	Description	Decimal Value	Hex Value
CWR	Congestion Window Reduced (CWR) flag is set by the sending host to shows that it received a TCP segment with the ECE flag set	128	80
ECE	ECN-Echo indicate that the TCP peer is ECN capable during 3-way handshake	64	40
URG	Indicates that the urgent pointer field is significant in this segment.	32	20
ACK	Indicates that the acknowledgment field is significant in this segment.	16	10
PSH	Push function to transfer data	08	08
RST	Resets the connection.	04	04
SYN	Synchronizes the sequence numbers.	02	02
FIN	Last packet from sender which means there is no more data.	01	01
NS	Nonce Sum flag used for concealment protection.	00	00

From given below image you can observe Hexadecimal information of TCP header field and using the given table you can study this value to obtain their original value.

Sequence and acknowledgment numbers are a major part of TCP, and they act as a way to guarantee that all data is transmitted consistently since all data transferred through a TCP connection must be acknowledged by the receiver in a suitable way. When an acknowledgment is not received, then the sender will again send all data that is unacknowledged.

TCP Header	Bits Color	Hex Value	Decimal value
Source Port	Pink	<u>ed f2</u>	60914
Destination Port (HTTP)	Lemon Yellow	00 50	80
Sequence Number	Dark Brown	8c 44 36 e7	2353280743
Acknowledgment Number	Grey	00 00 00 00	0
Flag (SYN)	Dark Yellow	02	02
Window size	Green	72 10	29,200
Checksum	Orange	84 4c	33,868
Urgent Pointer	Light Brown	00 00	00
Options	Red	*	*

```

▶ Frame 17: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interfa
▶ Ethernet II, Src: Vmware_d1:8e:0c (00:0c:29:d1:8e:0c), Dst: Giga-Byt_6a:9a:a2
▶ Internet Protocol Version 4, Src: 192.168.1.104, Dst: 192.168.1.101
▶ Transmission Control Protocol, Src Port: 60914, Dst Port: 80, Seq: layer 4 : 0

```

www.hackingarticles.in

layer 4 Hexa decimal data

Using given below table you can read Hex value of other Port Number and their Protocol services. Although these services operate after getting acknowledgment from the destination network and explore at application layer OSI model.

In this way, you can examine every layer of Wireshark for Network Packet Forensic.

Ports Number	Services	Hex Value	Decimal Value
21	FTP	15	21
22	SSH	16	22
23	Telnet	17	23
25	SMTP	19	25
53	DNS	35	53
80	HTTP	50	80

Password Sniffing

Because clear text protocols do not encrypt communication, all data, including passwords, is visible to the naked eye. Anyone who is in a position to see the communication (for example, a man in the middle) can eventually see everything.

In the sections that follow, we'll take a closer look at these protocols and see examples of captured passwords using Wireshark.

Capture HTTP Password

No introduction is certainly needed for the Hypertext Transfer Protocol (HTTP). It usually works on port 80/TCP, and as it is a text protocol, it does not give the communication parties much or no privacy. Anyone who's able to communicate can catch everything, including passwords, via that channel.

While all major browser vendors have made considerable efforts to prevent the use of HTTP as far as possible, during penetration testing, HTTP can be used on internal media.

Here is an example of login credentials captured in a POST request in an HTTP communication:

The image displays a Wireshark packet capture of an HTTP POST request. The main packet list shows a POST request to /userinfo.php from 192.168.0.107 to 18.192.172.30. The packet details pane shows the request headers and body. The body is an HTML Form URL Encoded data containing the username 'vijaymehta' and password 'maxelladiviner'.

Source	Destination	Protocol	Length	Info
192.168.0.100	224.0.0.251	MDNS	152	Standard query 0x0041 PTR _%9E5E7C8F47989526C9BCD95D24084F6F0B27C
192.168.0.104	224.0.0.251	MDNS	437	Standard query response 0x0000 PTR Y-Series-896e8e3a4462f9a459a9f
192.168.0.104	224.0.0.251	MDNS	404	Standard query response 0x0000 PTR Y-Series-896e8e3a4462f9a459a9f
192.168.0.104	224.0.0.251	MDNS	389	Standard query response 0x0000 PTR Y-Series-896e8e3a4462f9a459a9f
192.168.0.1	224.0.0.1	IGMPv3	60	Membership Query, general
192.168.0.107	18.192.172.30	TCP	74	52356 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=
18.192.172.30	192.168.0.107	TCP	74	80 → 52356 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1440 SACK_P
192.168.0.107	18.192.172.30	TCP	66	52356 → 80 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=2163476701 TS
192.168.0.107	18.192.172.30	HTTP	605	POST /userinfo.php HTTP/1.1 (application/x-www-form-urlencoded)
18.192.172.30	192.168.0.107	TCP	66	80 → 52356 [ACK] Seq=1 Ack=540 Win=30208 Len=0 TSval=773377626 TS
18.192.172.30	192.168.0.107	HTTP	342	HTTP/1.1 302 Found (text/html)
192.168.0.107	18.192.172.30	TCP	66	52356 → 80 [ACK] Seq=540 Ack=277 Win=64128 Len=0 TSval=2163476852
192.168.0.107	18.192.172.30	HTTP	460	GET /login.php HTTP/1.1

Wireshark · Packet 402 · http password.pcapng

Accept-Language: en-US,en;q=0.5\r\n
Accept-Encoding: gzip, deflate\r\n
Content-Type: application/x-www-form-urlencoded\r\n
> Content-Length: 36\r\n
Origin: http://testphp.vulnweb.com\r\n
Connection: keep-alive\r\n
Referer: http://testphp.vulnweb.com/login.php\r\n
Upgrade-Insecure-Requests: 1\r\n
\r\n
[Full request URI: http://testphp.vulnweb.com/userinfo.php]
[HTTP request 1/2]
[Response in frame: 404]
[Next request in frame: 406]
File Data: 36 bytes

HTML Form URL Encoded: application/x-www-form-urlencoded

- Form item: "uname" = "vijaymehta"
 - Key: uname
 - Value: vijaymehta
- Form item: "pass" = "maxelladiviner"
 - Key: pass
 - Value: maxelladiviner

Monitoring HTTPS packets over SSL or TLS

Dissect HTTPS Packet Captures

Open the provided HTTPS/TLS.pcapng file. Where you can see

- The 3-way handshake is happening
- Hello from SSL Client and the ACK from server
- Server Hello and then ACK
- Exchanging some key and Cipher information
- Started Exchanging Data

Wireshark packet capture of an HTTPS/TLS session. The packet list shows a 3-way handshake (SYN, SYN-ACK, ACK) followed by SSLv2 Client Hello, TCP ACK, SSLv3 Server Hello, Certificate, Server Hello Done, TCP ACK, SSLv3 Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message, and SSLv3 Application Data. The packet details pane shows the structure of the Application Data packet (503 bytes on wire, 503 bytes captured). The packet bytes pane shows the raw hex and ASCII data.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	127.0.0.1	127.0.0.1	TCP	74	38713 → 443 [SYN] Seq=0 Win=32767 Len=0 MSS=16396 SACK
2	0.000021	127.0.0.1	127.0.0.1	TCP	74	443 → 38713 [SYN, ACK] Seq=0 Ack=1 Win=32767 Len=0 MS
3	0.000037	127.0.0.1	127.0.0.1	TCP	66	38713 → 443 [ACK] Seq=1 Ack=1 Win=32767 Len=0 TSval=5
4	0.000158	127.0.0.1	127.0.0.1	SSLv2	171	Client Hello
5	0.000178	127.0.0.1	127.0.0.1	TCP	66	443 → 38713 [ACK] Seq=1 Ack=106 Win=32767 Len=0 TSval
6	0.002160	127.0.0.1	127.0.0.1	SSLv3	995	Server Hello, Certificate, Server Hello Done
7	0.002609	127.0.0.1	127.0.0.1	TCP	66	38713 → 443 [ACK] Seq=106 Ack=930 Win=32767 Len=0 TSv
8	2.808933	127.0.0.1	127.0.0.1	SSLv3	278	Client Key Exchange, Change Cipher Spec, Encrypted Ha
9	2.822770	127.0.0.1	127.0.0.1	SSLv3	141	Change Cipher Spec, Encrypted Handshake Message
10	2.822809	127.0.0.1	127.0.0.1	TCP	66	38713 → 443 [ACK] Seq=318 Ack=1005 Win=32767 Len=0 TS
11	2.833071	127.0.0.1	127.0.0.1	SSLv3	503	Application Data
12	2.873275	127.0.0.1	127.0.0.1	TCP	66	443 → 38713 [ACK] Seq=1005 Ack=755 Win=32767 Len=0 TS
13	2.938485	127.0.0.1	127.0.0.1	SSLv3	103	Encrypted Handshake Message
14	2.938750	127.0.0.1	127.0.0.1	SSLv3	183	Encrypted Handshake Message
15	2.938761	127.0.0.1	127.0.0.1	TCP	66	443 → 38713 [ACK] Seq=1042 Ack=872 Win=32767 Len=0 TS
16	2.938999	127.0.0.1	127.0.0.1	SSLv3	1073	Encrypted Handshake Message, Encrypted Handshake Mess
17	2.940026	127.0.0.1	127.0.0.1	SSLv3	337	Encrypted Handshake Message, Change Cipher Spec, Enchr
18	2.943406	127.0.0.1	127.0.0.1	SSLv3	172	Change Cipher Spec, Encrypted Handshake Message
19	2.944825	127.0.0.1	127.0.0.1	SSLv3	5756	Application Data, Application Data

Frame 11: 503 bytes on wire (4024 bits), 503 bytes captured (4024 bits)
 Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
 Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
 Transmission Control Protocol, Src Port: 38713, Dst Port: 443, Seq: 318, Ack: 1005, Len: 437
 Transport Layer Security

0000 00 00 00 00 00 00 00 00 00 00 00 08 00 45 00E.
 0010 01 e9 49 72 40 00 00 06 f1 9a 7f 00 00 01 7f 00 ..Ir@.....
 0020 00 01 97 39 01 bb 78 8c 3a d4 78 c5 28 c5 80 18 ...9...x:..x:(...
 0030 7f ff ff dd 00 00 01 01 08 0a 1f 53 7c 14 1f 53S|..S
 0040 7c 0a 17 03 00 01 b0 4a c3 3e 9d 77 78 01 2c b4 |.....J->wx.,
 0050 bc 4c 9a 84 d7 b9 90 0c 21 10 f0 fa 00 7c 16 bb ..L.....!.....
 0060 7f fb 72 42 4f ad 50 4a d0 aa 6f aa 44 6c 62 94 w.rB0PJ..o.Dlb
 0070 1b c5 fe e9 1c 5e de 85 0b 0e 05 e4 18 6e d2 d3^.....n..

Then, if we click on any application data, that data is unreadable to us. However, with Wireshark, we can decrypt that data... all we need is the server's Private Key. **Don't worry we have already provided the key along with the PCAP file.**

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

Apply a display filter ... <Ctrl-/>

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	127.0.0.1	127.0.0.1	TCP	74	38713 → 443 [SYN] Seq=0 Win=32767 Len=4
2	0.000021	127.0.0.1	127.0.0.1	TCP	74	443 → 38713 [SYN, ACK] Seq=0 Ack=1 Win=
3	0.000037	127.0.0.1	127.0.0.1	TCP	66	38713 → 443 [ACK] Seq=1 Ack=1 Win=32767
4	0.000158	127.0.0.1	127.0.0.1	SSLv2	171	Client Hello
5	0.000178	127.0.0.1	127.0.0.1	TCP	66	443 → 38713 [ACK] Seq=1 Ack=106 Win=327
6	0.002160	127.0.0.1	127.0.0.1	SSLv3	995	Server Hello, Certificate, Server Hello
7	0.002609	127.0.0.1	127.0.0.1	TCP	66	38713 → 443 [ACK] Seq=106 Ack=930 Win=
8	2.808933	127.0.0.1	127.0.0.1	SSLv3	278	Client Key Exchange, Change Cipher Spec
9	2.822770	127.0.0.1	127.0.0.1	SSLv3	141	Change Cipher Spec, Encrypted Handshake
10	2.822809	127.0.0.1	127.0.0.1	TCP	66	38713 → 443 [ACK] Seq=318 Ack=1005 Win=
11	2.833071	127.0.0.1	127.0.0.1	SSLv3	503	Application Data
12	2.873275	127.0.0.1	127.0.0.1	TCP	66	443 → 38713 [ACK] Seq=1005 Ack=755 Win=
13	2.938485	127.0.0.1	127.0.0.1	SSLv3	103	Encrypted Handshake Message
14	2.938750	127.0.0.1	127.0.0.1	SSLv3	183	Encrypted Handshake Message
15	2.938761	127.0.0.1	127.0.0.1	TCP	66	443 → 38713 [ACK] Seq=1042 Ack=872 Win=
16	2.938999	127.0.0.1	127.0.0.1	SSLv3	1073	Encrypted Handshake Message, Encrypted
17	2.940026	127.0.0.1	127.0.0.1	SSLv3	337	Encrypted Handshake Message, Change Cip
18	2.943406	127.0.0.1	127.0.0.1	SSLv3	172	Change Cipher Spec, Encrypted Handshake
19	2.944825	127.0.0.1	127.0.0.1	SSLv3	5756	Application Data, Application Data

<

> Frame 11: 503 bytes on wire (4024 bits), 503 bytes captured (4024 bits)

> Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)

> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1

> Transmission Control Protocol, Src Port: 38713, Dst Port: 443, Seq: 318, Ack: 1005, Len: 437

▼ Transport Layer Security

▼ SSLv3 Record Layer: Application Data Protocol: Application Data

Content Type: Application Data (23)

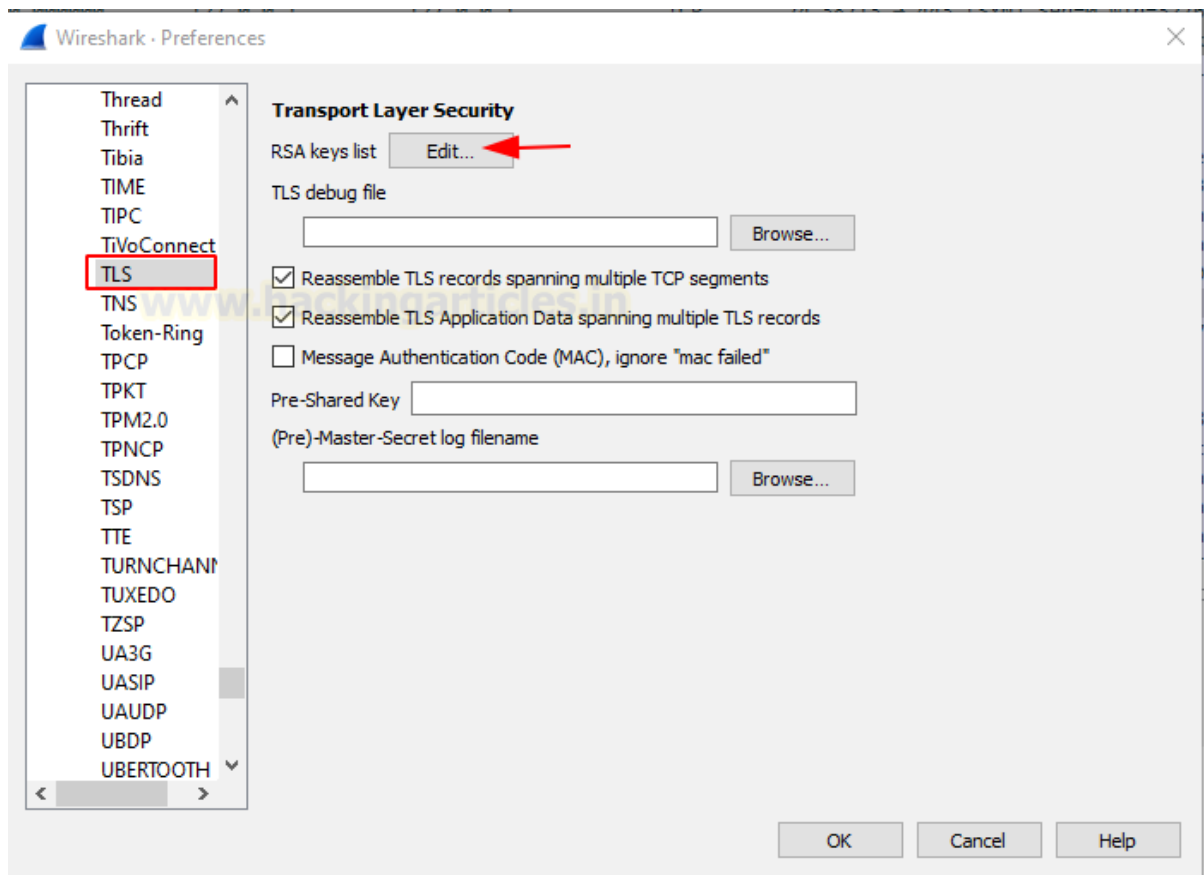
Version: SSL 3.0 (0x0300)

Length: 432

Encrypted Application Data: 4ac33e9d7778012cb4bc4c9a84d7b9900c2110f0fa007c16bb77fb72424fad504ad0aa6f...

To Decrypt the Encrypted Application Data over TLS or SSL Navigate to

Edit > Preference > Protocol > TLS

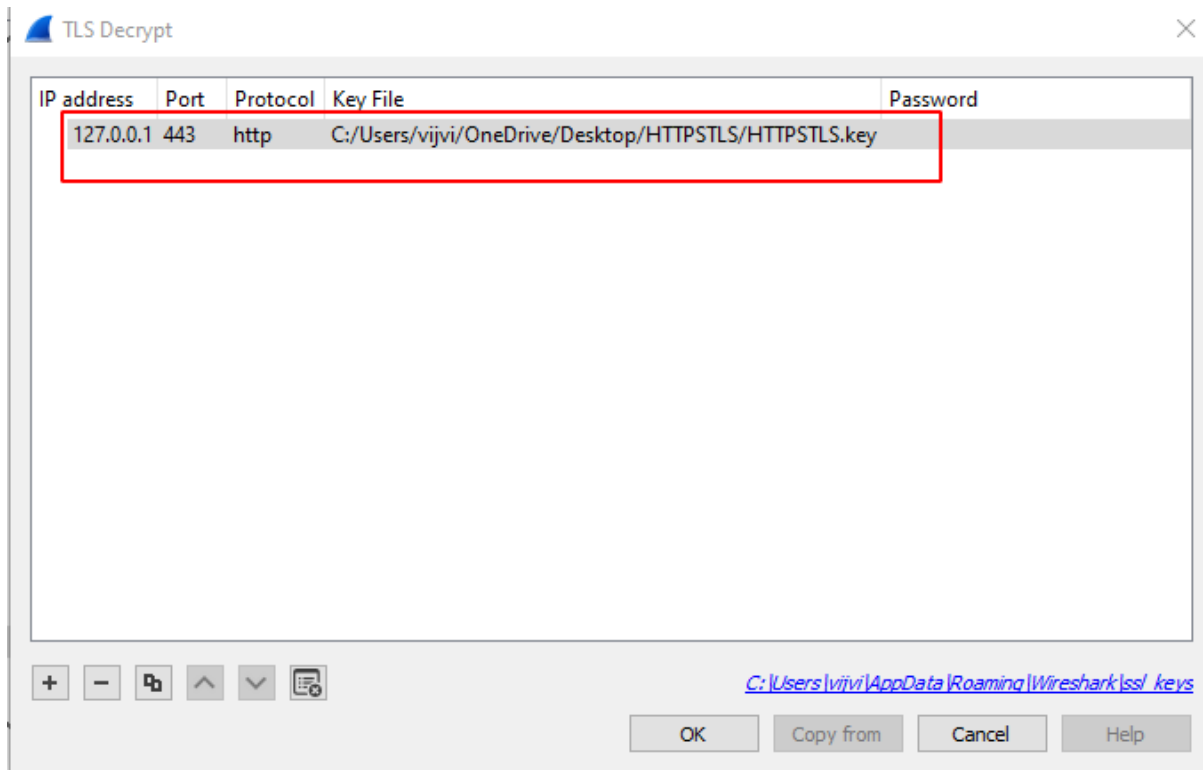


And add these values

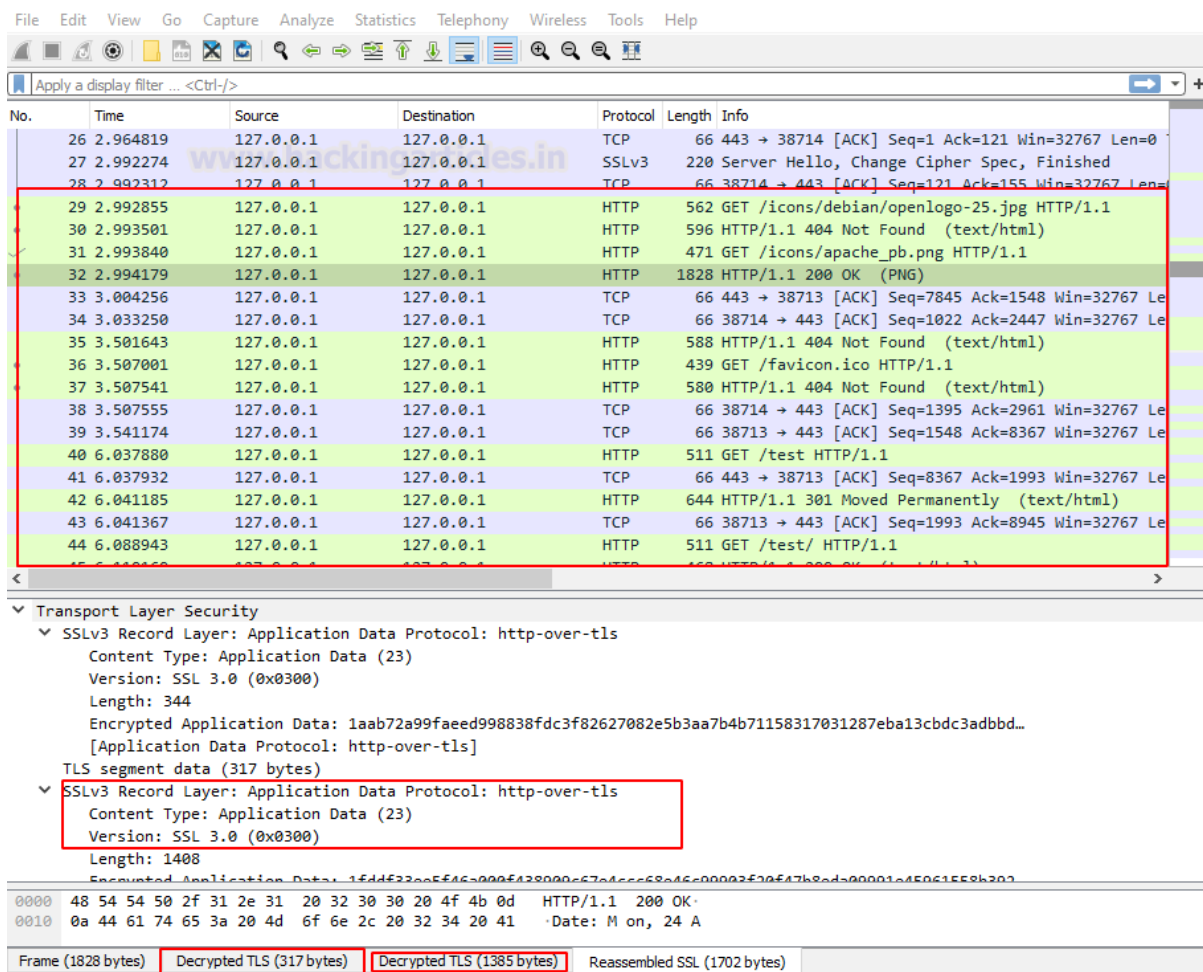
IP address: 127.0.0.1

Port: 443

Key File:



Hurray!!! As you can see, we have Successfully decrypted the Data over the TLS.



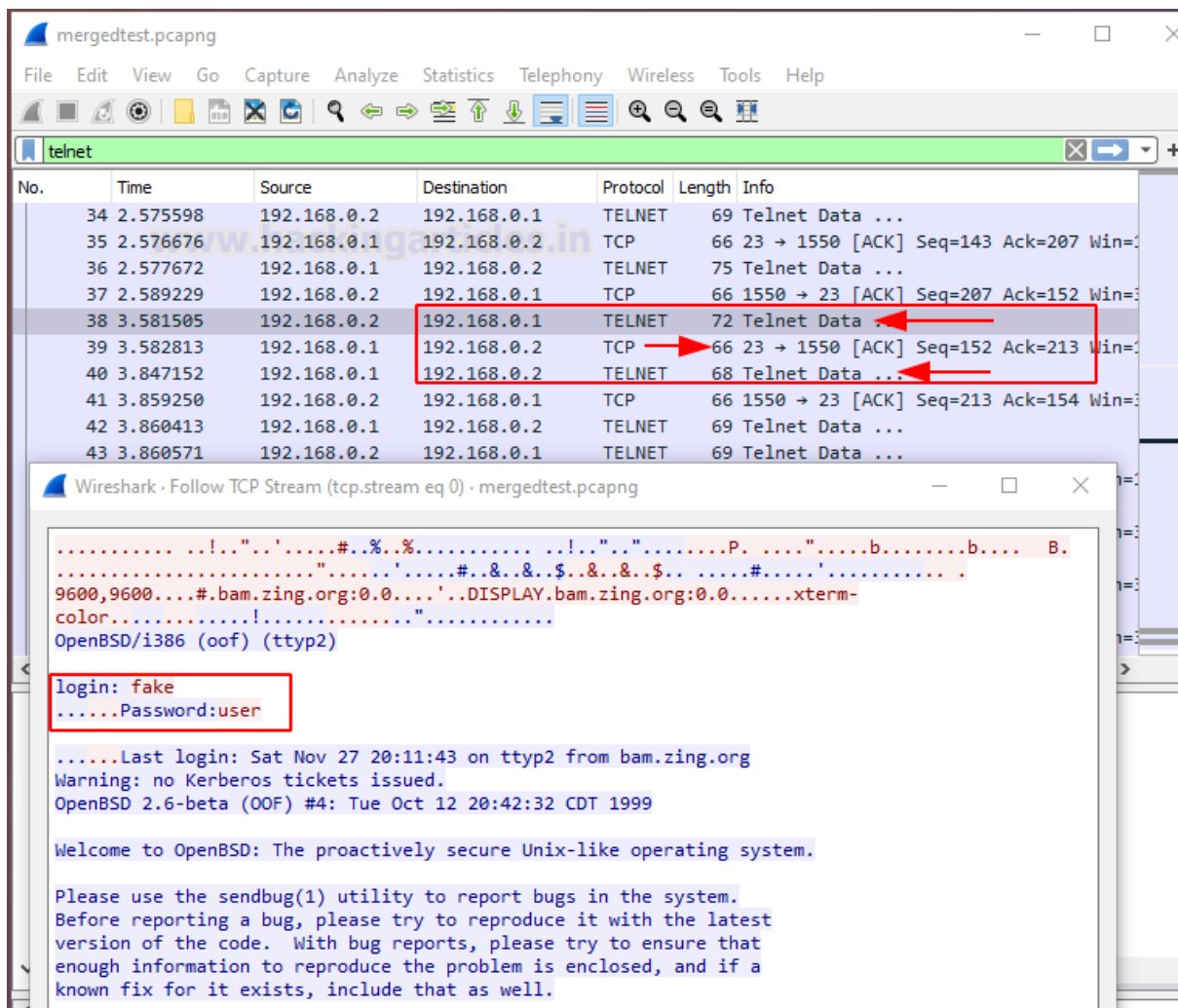
Capture Telnet Password

No introduction is required for Telnet protocol using port tcp/23. It is mainly used for administrative convenience and is known for its insecurity. Since encryption is not available, privacy or unauthorized access protection is not available. Telnet is still used today, however...

Telnet is a protocol used for administration on a wide range of devices. Telnet is the only option for some devices, with no other options (e.g. there is no SSH nor HTTPS web interface available). This makes it extremely difficult for organizations to completely eliminate it. Telnet is commonly seen on:

- Video Conferencing Systems
- Mainframes
- Network equipment
- Storage and Tape systems
- Imaging devices
- Legacy IP based Phones

Since telnet is a plain-text protocol, an opponent can wake up to the communication and capture it all, including passwords. The following screenshot shows an example of a telnet communication with the captured password:



So, that now you can see an attacker completely overtake the Mainframe System.

Capture FTP Password

File Transfer Protocol (FTP) usually uses the TCP/20 or the TCP/21 ports. Although this protocol is very old, it is still used in their networks by some organizations. FTP is a plain text protocol so a well-positioned attacker can capture FTP login credentials with Wireshark very easily. This screenshot shows a captured FTP password with Wireshark as an example:

ftp.pcap

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

tcp.stream eq 0

No.	Time	Source	Destination	Protocol	Length	Info
9	0.788801	10.0.1.100	198.145.20.140	TCP	78	55919 → 21 [SYN] Seq=0 Win=65535 Len=0 MSS=14
10	0.901830	198.145.20.140	10.0.1.100	TCP	74	21 → 55919 [SYN, ACK] Seq=0 Ack=1 Win=14480 L
11	0.902022	10.0.1.100	198.145.20.140	TCP	66	55919 → 21 [ACK] Seq=1 Ack=1 Win=132480 Len=0
12	1.020035	198.145.20.140	10.0.1.100	FTP	93	Response: 220 Welcome to kernel.org
13	1.020201	10.0.1.100	198.145.20.140	TCP	66	55919 → 21 [ACK] Seq=1 Ack=28 Win=132448 Len=
14	12.651747	10.0.1.100	198.145.20.140	FTP	76	Request: USER ftp
15	12.784279	198.145.20.140	10.0.1.100	TCP	66	21 → 55919 [ACK] Seq=28 Ack=11 Win=14592 Len=
16	12.784290	198.145.20.140	10.0.1.100	FTP	100	Response: 331 Please specify the password.
17	12.784481	10.0.1.100	198.145.20.140	TCP	66	55919 → 21 [ACK] Seq=11 Ack=62 Win=132416 Len=
18	19.293792	10.0.1.100	198.145.20.140	FTP	88	Request: PASS jowens@yccc.edu
19	19.400899	198.145.20.140	10.0.1.100	FTP	89	Response: 230 Login successful.
20	19.401116	10.0.1.100	198.145.20.140	TCP	66	55919 → 21 [ACK] Seq=33 Ack=85 Win=132392 Len=
21	19.401370	10.0.1.100	198.145.20.140	FTP	72	Request: SYST
22	19.523265	198.145.20.140	10.0.1.100	FTP	85	Response: 215 UNIX Type: L8
23	19.523444	10.0.1.100	198.145.20.140	TCP	66	55919 → 21 [ACK] Seq=39 Ack=104 Win=132376 Le
24	19.523671	10.0.1.100	198.145.20.140	FTP	72	Request: FEAT
25	19.615748	198.145.20.140	10.0.1.100	FTP	81	Response: 211-Features:

<

> Frame 18: 88 bytes on wire (704 bits), 88 bytes captured (704 bits)

> Ethernet II, Src: Apple_23:15:a4 (f0:b4:79:23:15:a4), Dst: Cisco-Li_fe:bd:f9 (00:23:69:fe:bd:f9)

> Internet Protocol Version 4, Src: 10.0.1.100, Dst: 198.145.20.140

> Transmission Control Protocol, Src Port: 55919, Dst Port: 21, Seq: 11, Ack: 62, Len: 22

▼ File Transfer Protocol (FTP)

▼ PASS jowens@yccc.edu\r\n

Request command: PASS

Request arg: jowens@yccc.edu

[Current working directory:]

As you can see by sitting in a network, we can easily capture FTP credentials.

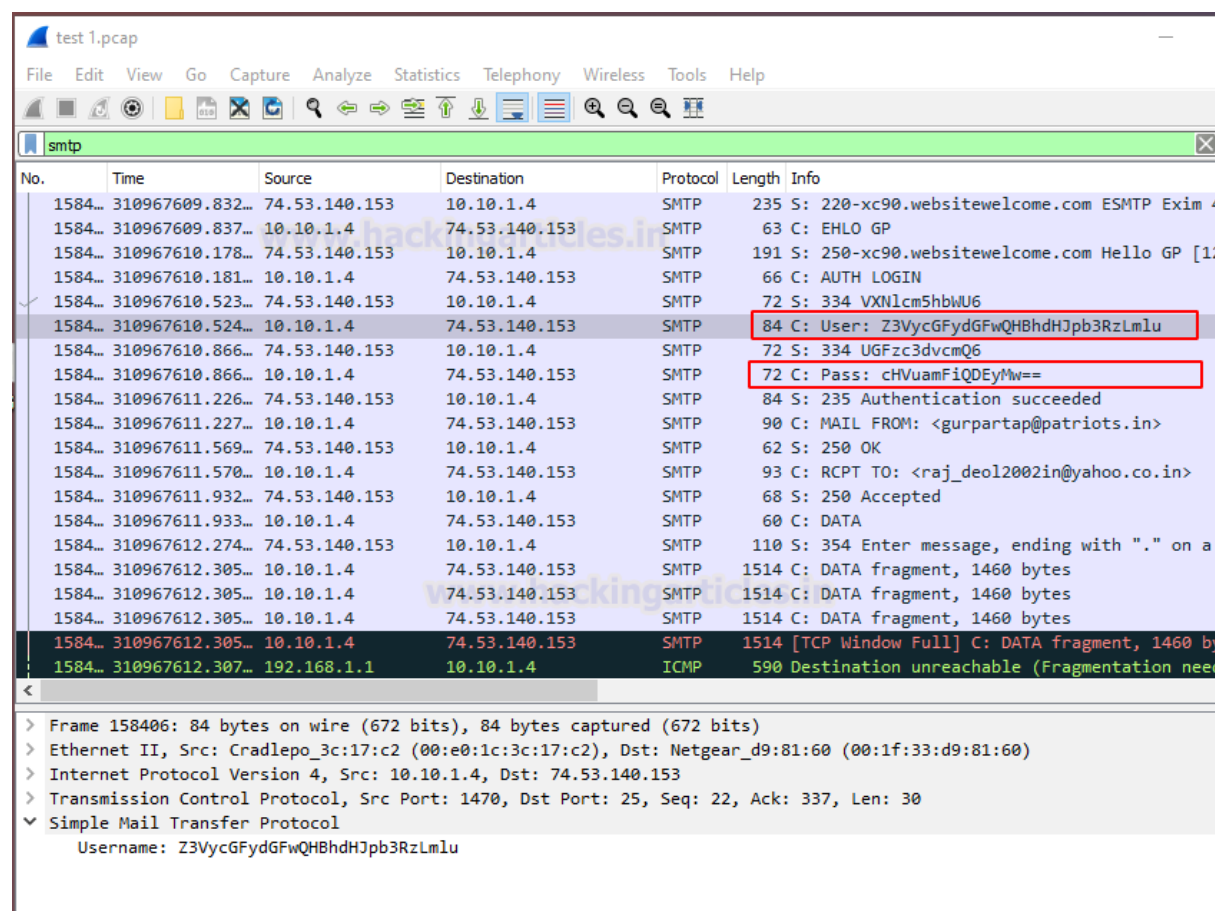
Capture SMTP password

For many decades, we have also been accompanied by SMTP (Simple Mail Transfer Protocol). It uses TCP/25 and although the port TCP/464 is secure, today the port TCP/25 is almost opened on each mail server because of reverse compatibility.

Many TCP/25 servers need the command 'STARTTLS' to begin the encryption of SSL/TLS before any attempts are made to authenticate it. However, mail servers still support plain text authentication across the unencrypted channel within certain organizations. Mostly because of heritage systems in your internal networks.

If someone is using plain text authentication during an SMTP transaction, the credentials can be sniffed from a well-positioned attacker. The attacker must only decode the username and password from base64. SMTP uses Base 64 encoding for the transaction to encode the username and password.

A captured SMTP credentials can be seen in the following screenshot with Wireshark and the consequent base64 decoder using the base64 utility.



There are many methods available to decode the base64 strings. For this, I'm using an online tool that is designed specifically for decoding such as base64decode.org or base64decode.net. But we should beware – we may not want to disclose private credentials on the Internet to other parties. In the course of penetration tests and offensive tests, sensitivity and privacy are especially crucial. This is particularly important.

Now, just copy the value of strings of user and password and decode it via base64 decoder as shown below image. As of now, I'm decrypting the user string

BASE64

Decode and Encode

Decode Encode

Do you have to deal with **Base64** format? Then this site is perfect for you! Use our super handy online tool to encode or **decode** your data.

Decode from Base64 format

Simply enter your data then push the decode button.

Z3VycGFydGFwQHBhdHJpb3RzLmlu

i For encoded binaries (like images, documents, etc.) use the file upload form a little further down on this page.

UTF-8 Source character set.

☐ Decode each line separately (useful for when you have multiple entries).

☒ Live mode OFF Decodes in real-time as you type or paste (supports only the UTF-8 character set).

< DECODE > Decodes your data into the area below.

gurpartap@patriots.in

User: – Z3VycGFydGFwQHBhdHJpb3RzLmlu

As you can see in the above screenshot, we have successfully able to see the user's name in clear text format. Similarly, we can decrypt the password

Password: – cHVuamFiQDEyMw==

BASE64

Decode

Encode

Decode and Encode

Do you have to deal with **Base64** format? Then this site is perfect for you! Use our super handy online tool to encode or **decode** your data.

Decode from Base64 format

Simply enter your data then push the decode button.

cHVuamFiQDEyMw==

For encoded binaries (like images, documents, etc.) use the file upload form a little further down on this page.

UTF-8

Source character set.

☐

 Decode each line separately (useful for when you have multiple entries).

Live mode OFF

 Decodes in real-time as you type or paste (supports only the UTF-8 character set).

< **DECODE** >

 Decodes your data into the area below.

punjab@123

Hurray!!! Now we have got enough credentials to take over a system.

Analyzing SNMP Community String

Simple Network Management Protocol (SNMP) typically runs on port UDP/161. The main objective is network devices and their functions to manage and monitor. SNMP have 3 versions and the first 2 (v1 and v2c) versions are plain text. SNMP uses something that is equivalent to authentication, named community string. Therefore, it is almost the same to capture the SNMP community string as to capture credentials.

While SNMPv3 has been with us for nearly two decades, it takes time. In their internal networks, most organizations still use v1 or v2c. Typically this is due to the backwards compatibility in their networks with legacy systems.

An example of the SNMP community string captured using Wireshark is:

The image shows a Wireshark packet capture of an SNMPv1 trap. The packet list shows a trap message from 192.168.0.2 to 192.168.0.1. The packet details pane shows the structure of the trap, including the community string 'public'. The packet bytes pane shows the raw data.

No.	Time	Source	Destination	Protocol	Length	Info
15262	620.143771	192.168.0.2	192.168.0.1	SNMP	99	iso.3.6.1.4.1.4.1.2.21[Ma]
15263	620.183662	192.168.0.2	192.168.0.1	SNMP	98	iso.3.6.1.4.1.4.1.2.21[Ma]
15264	620.223650	192.168.0.2	192.168.0.1	SNMP	92	trap iso.3.6.1.4.1.4.1.2.21
15265	620.263653	192.168.0.2	192.168.0.1	SNMP	100	trap iso.3.6.1.4.1.4.1.2.21
15266	620.303646	192.168.0.2	192.168.0.1	SNMP	100	trap iso.3.6.1.4.1.4.1.2.21
15267	620.343642	192.168.0.2	192.168.0.1	SNMP	100	iso.3.6.1.4.1.4.1.2.21 1.3
15268	620.383630	192.168.0.2	192.168.0.1	SNMP	100	trap iso.3.6.1.4.1.4.1.2.21
15269	620.423625	192.168.0.2	192.168.0.1	SNMP	100	trap iso.3.6.1.4.1.4.1.2.21
15270	620.463627	192.168.0.2	192.168.0.1	SNMP	100	trap iso.3.6.1.4.1.4.1.2.21
15271	620.503618	192.168.0.2	192.168.0.1	SNMP	100	trap iso.3.6.1.4.1.4.1.2.21
15272	620.543611	192.168.0.2	192.168.0.1	SNMP	102	trap iso.3.6.1.4.1.4.1.2.21
15273	620.583613	192.168.0.2	192.168.0.1	SNMP	86	trap iso.3.6.1.4.1.4.1.2.21
15274	620.623614	192.168.0.2	192.168.0.1	SNMP	86	trap iso.3.6.1.4.1.4.1.2.21
15275	620.663608	192.168.0.2	192.168.0.1	SNMP	154	trap iso.3.6.1.4.1.4.1.2.21
15276	620.703604	192.168.0.2	192.168.0.1	SNMP	225	trap iso.3.6.1.4.1.4.1.2.21
15277	620.743596	192.168.0.2	192.168.0.1	SNMP	259	trap iso.3.6.1.4.1.4.1.2.21
15278	620.783933	192.168.0.2	192.168.0.1	SNMP	432	trap iso.3.6.1.4.1.4.1.2.21
15279	620.823602	192.168.0.2	192.168.0.1	SNMP	636	trap iso.3.6.1.4.1.4.1.2.21

Simple Network Management Protocol

- version: version-1 (0)
- community: public
- data: trap (4)
 - trap
 - enterprise: 1.3.6.1.4.1.4.1.2.21 (iso.3.6.1.4.1.4.1.2.21)
 - agent-addr: 127.0.0.1
 - generic-trap: egpNeighborLoss (5)
 - specific-trap: 0
 - time-stamp: 15270
 - variable-bindings: 1 item
 - 1.3.6.1.2.1.2.1.0

0000 00 e0 29 68 8b fb 00 20 af 1b 07 fa 08 00 45 00 ..)h... ..E.
0010 00 56 4f 6c 00 00 40 11 a9 d7 c0 a8 00 02 c0 a8 .VOL..@.
0020 00 01 04 10 00 a2 00 42 9a 1f 30 38 02 01 00 04B ..08..
0030 06 70 75 62 6c 69 63 a4 2b 06 09 2b 06 01 04 01 .public. ++++..

An attacker could now use the community string and collect detailed system information. This could enable the attacker to learn about the system insensitive detail and to make further attempts. Note that the community string sometimes also allows you to modify your remote system configuration (read/write access).

Capture MSSQL Password

The Microsoft SQL server usually runs on TCP/1433 port; this is yet another service we can use with Wireshark to capture the password. If the server is not configured using the ForceEncryption option, it is possible to record plain text authentication directly or via a downgrade attack. MSSQL credentials can be easily captured by a man in the middle.

Here's an example of a Wireshark-captured MSSQL

The image shows a Wireshark network traffic capture. The top pane displays a list of packets. Packet 6, at time 0.000559, is a MySQL packet (length 132) from 192.168.0.254 to 192.168.0.254, labeled '132 Login Request user=tfoerste'. The bottom pane shows the details of this packet, expanded to the 'MySQL Protocol' section. Under 'Login Request', the 'Username' is 'tfoerste' and the 'Password' is 'eefd6d5562851bc5966a0b41236ae3f2315efcc4'. Both fields are highlighted with red arrows pointing to them.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.0.254	192.168.0.254	TCP	74	56162 → 3306 [SYN] Seq=0 Win=3279
2	0.000046	192.168.0.254	192.168.0.254	TCP	74	3306 → 56162 [SYN, ACK] Seq=0 Ack=
3	0.000077	192.168.0.254	192.168.0.254	TCP	66	56162 → 3306 [ACK] Seq=1 Ack=1 Wi
4	0.000265	192.168.0.254	192.168.0.254	MySQL	122	Server Greeting proto=10 version=
5	0.000286	192.168.0.254	192.168.0.254	TCP	66	56162 → 3306 [ACK] Seq=1 Ack=57 W
6	0.000559	192.168.0.254	192.168.0.254	MySQL	132	Login Request user=tfoerste
7	0.000583	192.168.0.254	192.168.0.254	TCP	66	3306 → 56162 [ACK] Seq=57 Ack=67
8	0.000695	192.168.0.254	192.168.0.254	MySQL	77	Response OK
9	0.000893	192.168.0.254	192.168.0.254	MySQL	103	Request Query
10	0.001051	192.168.0.254	192.168.0.254	MySQL	162	Response
11	0.040792	192.168.0.254	192.168.0.254	TCP	66	56162 → 3306 [ACK] Seq=104 Ack=16
12	5.698832	192.168.0.254	192.168.0.254	MySQL	88	Request Query
13	5.699011	192.168.0.254	192.168.0.254	MySQL	130	Response
14	5.699035	192.168.0.254	192.168.0.254	TCP	66	56162 → 3306 [ACK] Seq=126 Ack=22
15	5.699226	192.168.0.254	192.168.0.254	MySQL	75	Request Use Database
16	5.699324	192.168.0.254	192.168.0.254	MySQL	77	Response OK
17	5.699573	192.168.0.254	192.168.0.254	MySQL	85	Request Query
18	5.699998	192.168.0.254	192.168.0.254	MySQL	174	Response
19	5.700180	192.168.0.254	192.168.0.254	MySQL	82	Request Query
20	5.700418	192.168.0.254	192.168.0.254	MySQL	160	Response

> Transmission Control Protocol, Src Port: 56162, Dst Port: 3306, Seq: 1, Ack: 57, Len: 66

▼ MySQL Protocol

Packet Length: 62

Packet Number: 1

▼ Login Request

- > Client Capabilities: 0xa685
- > Extended Client Capabilities: 0x0003
- MAX Packet: 16777216
- Charset: utf8 COLLATE utf8_general_ci (33)
- Unused: 00
- Username: tfoerste
- Password: eefd6d5562851bc5966a0b41236ae3f2315efcc4

Now, we have a privileged account of the MSSQL server. Therefore, this would have a critical impact allowing the attacker to take complete control over the database server or it could also lead to remote command execution (RCE).

Capture PostgreSQL Password

PostgreSQL is yet another widely used SQL database server. It runs on TCP port 5432 and accepts a variety of authentication methods. It is usually set to disallow clear-text authentication, but it can also be set to allow it. In such cases, a well-positioned attacker could intercept network traffic and obtain the username and password.

It should be noted that PostgreSQL authentication occurs in multiple packets. The username and database name comes first:

The image shows a Wireshark packet capture of a PostgreSQL authentication sequence. The packet list pane shows several packets, with packet 9 (Time: 0.003085, Source: 127.0.0.1, Destination: 127.0.0.1, Protocol: PGSQL, Length: 104) highlighted with a red box. The packet details pane shows the PostgreSQL protocol structure, with the 'Parameter name: user' field highlighted by a red box and an arrow, and its value 'oryx' also highlighted with a red box. The 'Parameter name: database' field is 'mailstore'. The packet bytes pane shows the raw data in hexadecimal and ASCII.

No.	Time	Source	Destination	Protocol	Length	Info
7	0.002956	127.0.0.1	127.0.0.1	PGSQL	104	>
9	0.003085	127.0.0.1	127.0.0.1	PGSQL	104	>
11	0.003253	127.0.0.1	127.0.0.1	PGSQL	79	<R
13	0.003340	127.0.0.1	127.0.0.1	PGSQL	79	<R
15	0.003458	127.0.0.1	127.0.0.1	PGSQL	107	>p
16	0.003582	127.0.0.1	127.0.0.1	PGSQL	107	>p
19	0.077078	127.0.0.1	127.0.0.1	PGSQL	227	<R/S/S/S/S/S/K/Z
20	0.077986	127.0.0.1	127.0.0.1	PGSQL	227	<R/S/S/S/S/S/K/Z
21	0.078098	127.0.0.1	127.0.0.1	PGSQL	222	>P/B/D/E/S/P/B/D/E/S
23	0.078598	127.0.0.1	127.0.0.1	PGSQL	192	>P/B/E/P/B/D/E/S
25	0.080906	127.0.0.1	127.0.0.1	PGSQL	164	<1/2/C/1/2/T/D/C/Z
26	0.081378	127.0.0.1	127.0.0.1	PGSQL	118	>P/B/D/E/S
27	0.082503	127.0.0.1	127.0.0.1	PGSQL	99	<1/2/n/C/Z
28	0.083214	127.0.0.1	127.0.0.1	PGSQL	251	<1/2/T/C/Z
29	0.084898	127.0.0.1	127.0.0.1	PGSQL	145	<1/2/T/C/Z
32	8.949436	127.0.0.1	127.0.0.1	PGSQL	327	>P/B/D/E/S
34	9.099196	127.0.0.1	127.0.0.1	PGSQL	348	<1/2/T/C/Z
36	15.399924	127.0.0.1	127.0.0.1	PGSQL	114	>B/D/E/S

> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
> Transmission Control Protocol, Src Port: 45931, Dst Port: 5432, Seq: 1, Ack: 1, Len: 38
▼ PostgreSQL
 Type: Startup message
 Length: 38
 Protocol major version: 3
 Protocol minor version: 0
 Parameter name: user
 Parameter value: oryx
 Parameter name: database
 Parameter value: mailstore

0000 00 00 00 00 00 00 00 00 00 00 00 00 08 00 45 00E.
0010 00 5a bd 98 40 00 40 06 7f 03 7f 00 00 01 7f 00 ..Z..@..@.....
0020 00 01 b3 6b 15 38 c9 01 b0 bd c9 49 e2 1d 80 18 ...k.8...I..
0030 7f ff a0 48 00 00 01 01 08 0a 13 42 0d 2c 13 42 ...H....B.,B

PostgreSQL: Protocol Packets: 36

We can also see the PostgreSQL password in the following network packet:

The image shows a Wireshark network packet capture interface. The top menu bar includes File, Edit, View, Go, Capture, Analyze, Statistics, Telephony, Wireless, Tools, and Help. Below the menu is a toolbar with various icons. The main display area shows a list of network packets. Packet 15 is selected and highlighted with a red box. The packet details pane on the right shows the structure of the selected packet, with the PostgreSQL password message highlighted by a red box and a red arrow pointing to the password field.

No.	Time	Source	Destination	Protocol	Length	Info
7	0.002956	127.0.0.1	127.0.0.1	PGSQL	104	>
9	0.003085	127.0.0.1	127.0.0.1	PGSQL	104	>
11	0.003253	127.0.0.1	127.0.0.1	PGSQL	79	<R
13	0.003340	127.0.0.1	127.0.0.1	PGSQL	79	<R
15	0.003458	127.0.0.1	127.0.0.1	PGSQL	107	>p
16	0.003582	127.0.0.1	127.0.0.1	PGSQL	107	>p
19	0.077078	127.0.0.1	127.0.0.1	PGSQL	227	<R/S/S/S/S/S/K/Z
20	0.077986	127.0.0.1	127.0.0.1	PGSQL	227	<R/S/S/S/S/S/K/Z
21	0.078098	127.0.0.1	127.0.0.1	PGSQL	222	>P/B/D/E/S/P/B/D/E/S
23	0.078598	127.0.0.1	127.0.0.1	PGSQL	192	>P/B/E/P/B/D/E/S
25	0.080906	127.0.0.1	127.0.0.1	PGSQL	164	<1/2/C/1/2/T/D/C/Z
26	0.081378	127.0.0.1	127.0.0.1	PGSQL	118	>P/B/D/E/S
27	0.082503	127.0.0.1	127.0.0.1	PGSQL	99	<1/2/n/C/Z
28	0.083214	127.0.0.1	127.0.0.1	PGSQL	251	<1/2/T/C/Z
29	0.084898	127.0.0.1	127.0.0.1	PGSQL	145	<1/2/T/C/Z
32	8.949436	127.0.0.1	127.0.0.1	PGSQL	327	>P/B/D/E/S
34	9.099196	127.0.0.1	127.0.0.1	PGSQL	348	<1/2/T/C/Z
36	15.399924	127.0.0.1	127.0.0.1	PGSQL	114	>B/D/E/S

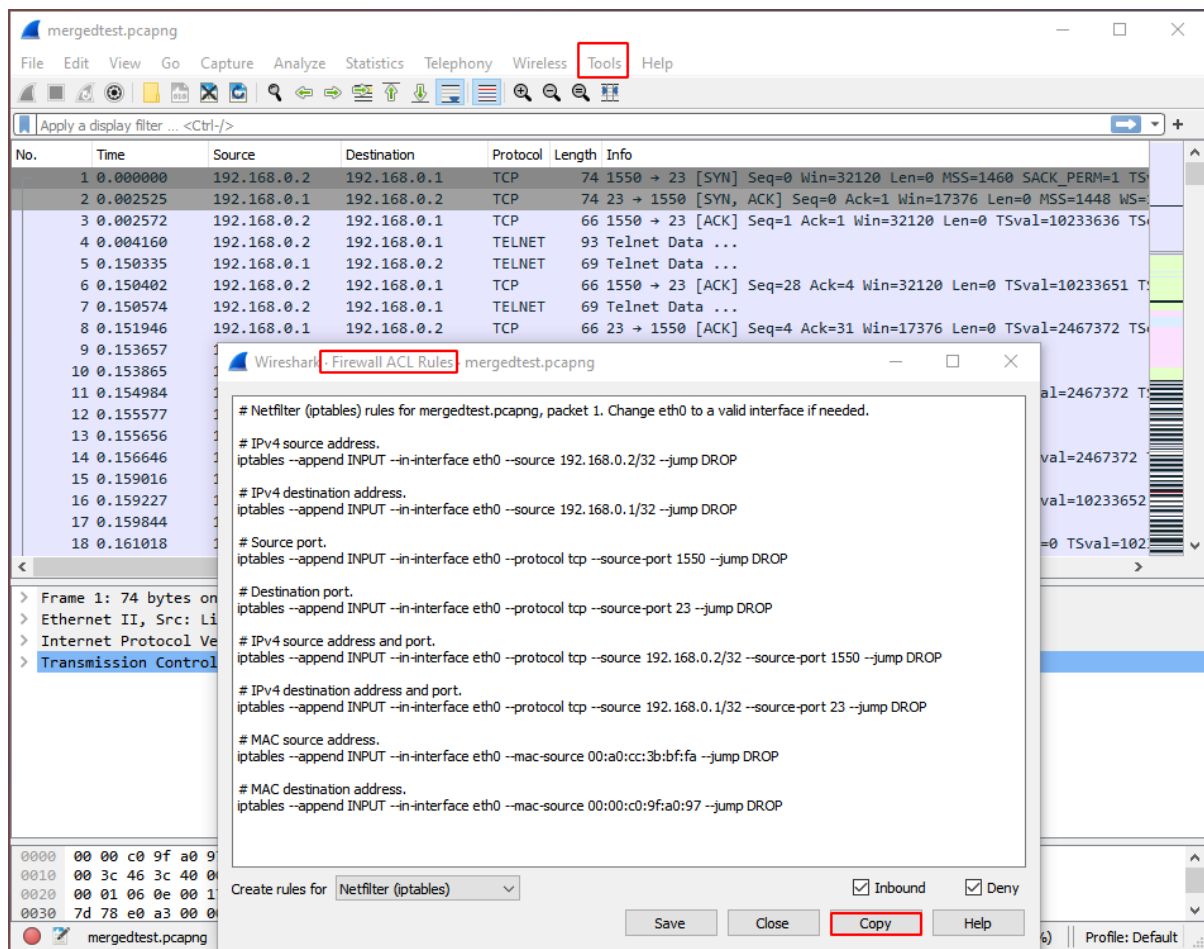
> Frame 15: 107 bytes on wire (856 bits), 107 bytes captured (856 bits)
> Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
> Transmission Control Protocol, Src Port: 45930, Dst Port: 5432, Seq: 39, Ack: 14, Len: 107
✓ PostgreSQL
 Type: Password message
 Length: 40
 Password: md5ceffc01dcde7541829deef6b5e9c9142

0040 0d 2c 70 00 00 00 28 6d 64 35 63 65 66 66 63 30 ·,p···(m d5ceffc0
0050 31 64 63 64 65 37 35 34 31 38 32 39 64 65 65 66 1dcde754 1829deef
0060 36 62 35 65 39 63 39 31 34 32 00 6b5e9c91 42·

A password. (pgsql.password), 36 bytes

Creating Firewall Rules with Wireshark

Although Wireshark cannot block network traffic, it can assist us in the development of firewall rules for our firewall. Wireshark will create firewall rules based on the traffic we're looking at. To block a packet, all we have to do is pick it and navigate through the menu:



Selected rules can now be copied and pasted directly into our firewall. The following firewalls' syntax is supported by Wireshark:

- Windows Firewall(netsh)
- IP Filter(ipfw)
- NetFilter (iptables)
- Packet Filter(pf)

Conclusion

Wireshark can catch authentication for a wide range of network protocols. There is a possibility as long as we have the ability to eavesdrop on network traffic and the communication is not encrypted. Passwords aren't the only thing that a well-placed attacker can capture; virtually any type of data passing through the network can be captured.

Hence, one can make use of these commands as a cybersecurity professional to assess vulnerabilities on systems and keep these systems away from threat.

References

- <https://www.hackingarticles.in/network-packet-forensic-using-wireshark/>
- <https://www.hackingarticles.in/wireshark-for-pentester-password-sniffing/>
- <https://www.wireshark.org/docs/>