INDEX

PART – A

1. Information gathering

A. Perform information gathering using nslookup and google dorking.

a. Identify the IPv4 address associated with a given domain name using an appropriate DNS query method.

nslookup –type=A <domain name>: is used to retrieve the IPv4 address of the given domain name by querying its A (Address) record in the DNS.

```
C:\Windows\System32>nslookup -type=A yenepoya.edu.in
Server:  UnKnown
Address:  fe80::a4c6:f0ff:fe06:5064

Non-authoritative answer:
Name:    yenepoya.edu.in
Address:  13.234.131.182
```

b. Retrieve information about the mail exchange server(s) responsible for handling email for a given domain.

nslookup –type=mx <domain name>: is used to find the Mail Exchange (MX) records for a specific domain. MX records indicate the mail servers responsible for receiving email on behalf of that domain.

```
C:\Windows\System32>nslookup -type=mx yenepoya.edu.in
Server:  UnKnown
Address:  fe80::a4c6:f0ff:fe06:5064

Non-authoritative answer:
yenepoya.edu.in MX preference = 0, mail exchanger = aspmx.l.google.com
yenepoya.edu.in MX preference = 10, mail exchanger = alt1.aspmx.l.google.com
yenepoya.edu.in MX preference = 10, mail exchanger = alt2.aspmx.l.google.com
yenepoya.edu.in MX preference = 20, mail exchanger = aspmx2.googlemail.com
yenepoya.edu.in MX preference = 20, mail exchanger = aspmx3.googlemail.com
yenepoya.edu.in MX preference = 30, mail exchanger = aspmx4.googlemail.com
yenepoya.edu.in MX preference = 30, mail exchanger = aspmx5.googlemail.com
```

c. Obtain the IPv6 address linked to a specific domain name using the correct DNS record type.

nslookup –type=AAAA <domain name>: This command is used to retrieve the IPv6 address of a domain by querying its AAAA (quad-A) record in DNS.

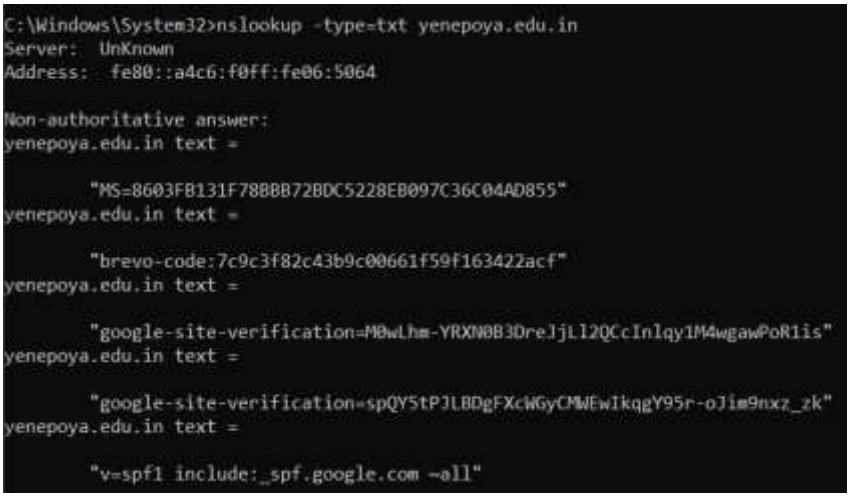
```
C:\Windows\System32>nslookup -type=AAAA yenepoya.edu.in
Server:  UnKnown
Address:  fe80::a4c6:f0ff:fe06:5064

Name:    yenepoya.edu.in
```

d. Extract human-readable and machine-readable text records for a domain.

nslookup –type=txt <domain name>: is used to query the TXT records of the domain yenepoya.edu.in.

TXT (Text) records in DNS are used to store arbitrary human-readable text. Common uses include:

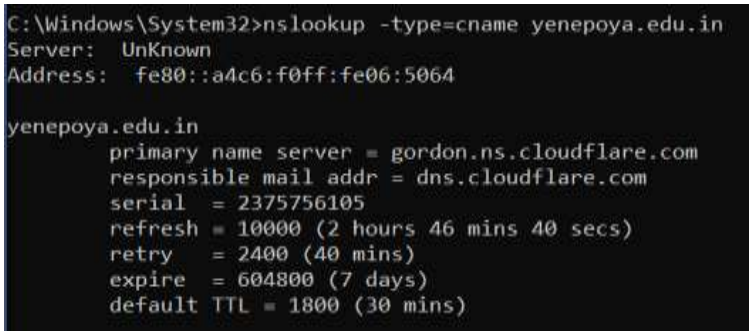
- SPF (Sender Policy Framework) – email spam prevention.
- DKIM (DomainKeys Identified Mail) – email authentication.
- DMARC (Domain-based Message Authentication, Reporting, and Conformance) – email spoofing protection.
- Google site verification, Microsoft domain verification, etc.



e. Discover the canonical (alias) name of a given domain by querying the appropriate DNS record type.

nslookup –type=cname <domain name>: is used to look up the CNAME (Canonical Name) record for a given domain.
www.example.com canonical name = example.com

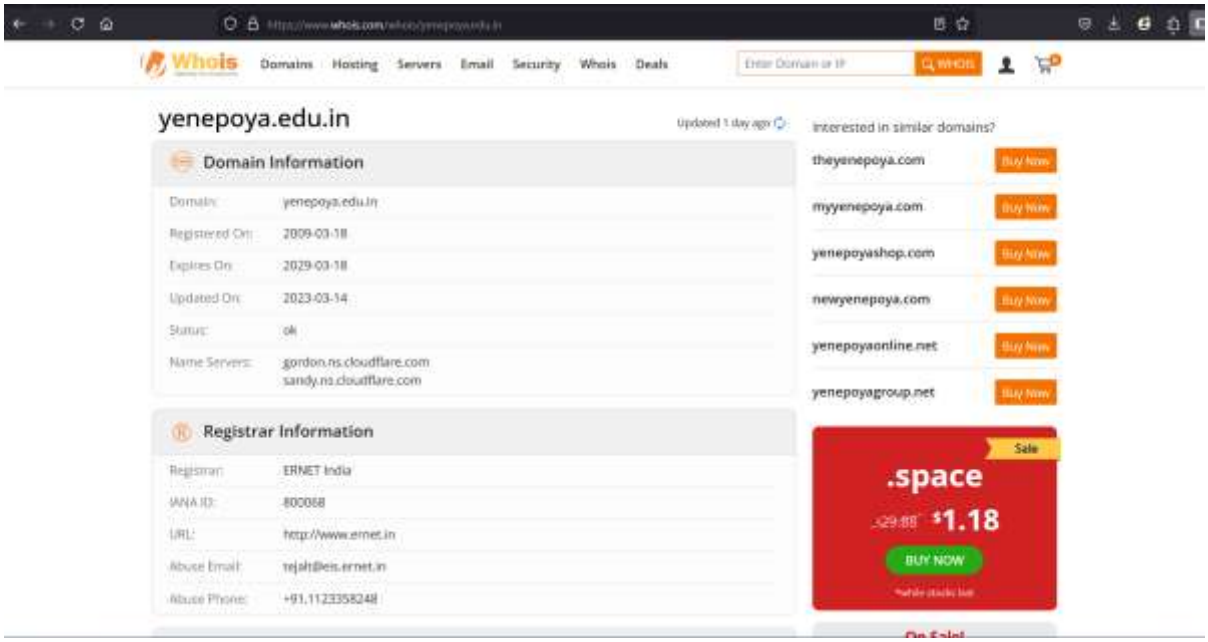
- When multiple subdomains point to the same domain (e.g., www, ftp, mail).
- To simplify domain management by pointing multiple aliases to one domain.
- Commonly used with CDNs (Content Delivery Networks), like cdn.example.com → realhost.cloudprovider.net.



B. Conduct a reconnaissance exercise on a given domain to gather technical, ownership, and historical information using open-source intelligence (OSINT) techniques.

a. whois

- The whois command is used to retrieve registration information about a domain name, IP address, or autonomous system (AS).
- It shows public records stored in domain registries and registrars, including:
- Domain owner information (name, organization, email, etc.)
- Registrar name (e.g., GoDaddy, Namecheap)
- Domain creation and expiration dates
- Name servers
- Status of the domain (e.g., active, clientTransferProhibited)
- Contact details (sometimes redacted for privacy)



b. sitereport netcraft

Netcraft Site Report is a free online tool provided by Netcraft that gives detailed technical information about a website or domain.

i. Hosting Details
- IP address
- Hosting provider
- Country of the hosting server

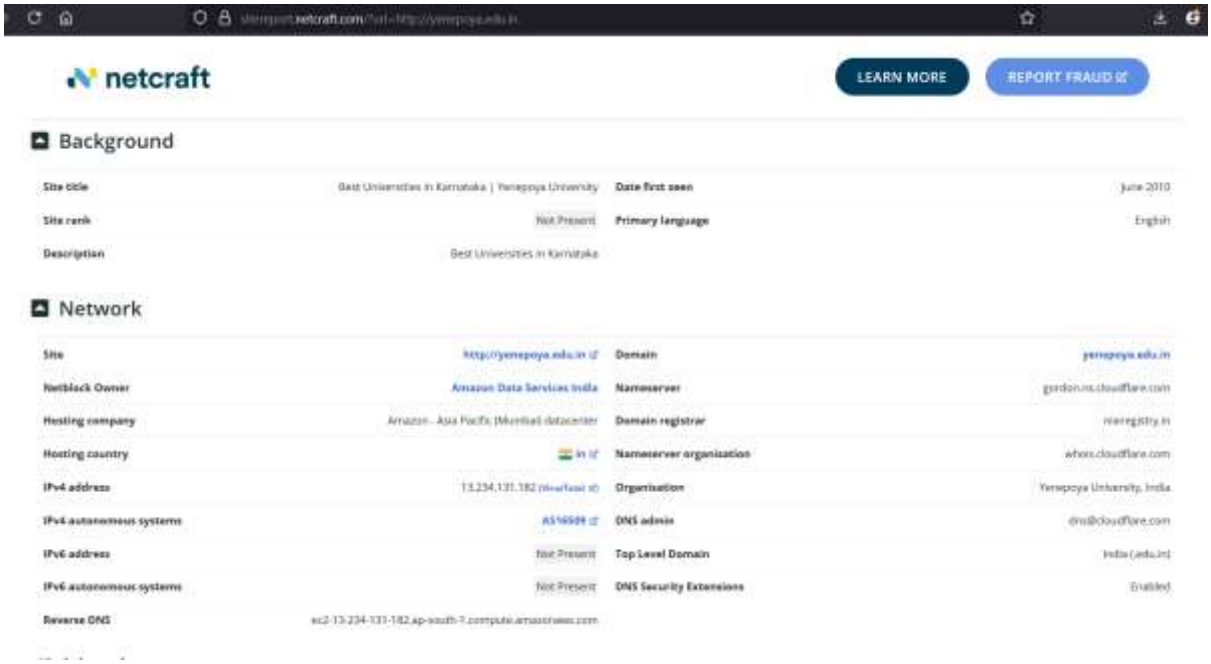
ii. Technology Stack
- Web server type (e.g., Apache, Nginx, Microsoft-IIS)
- Operating system
- Content Management System (CMS), if detected

iii. SSL/TLS Information
- HTTPS support
- Certificate issuer
- Expiry dates

iv. Site Rank & Popularity
- Site ranking (globally or within a country)
- Netcraft Risk Rating

v. Domain Registration Info
- Registrar
- Domain age
- WHOIS data link

vi. Security Insights
- Phishing site detection
- Risk score (helpful for assessing trustworthiness)



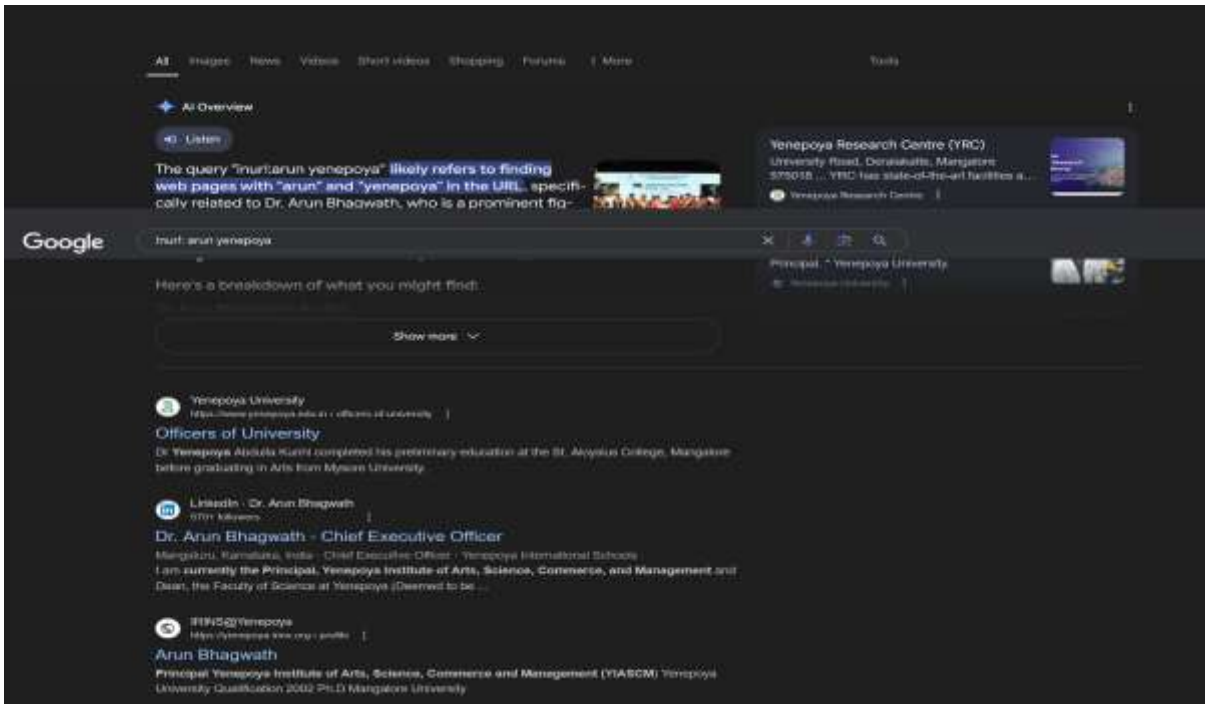
c. Google Dorking

i. inurl: <keyword> <domain name>

is a Google (or search engine) advanced search query used to find webpages where the keyword appears in the URL of a specific domain or set of pages.

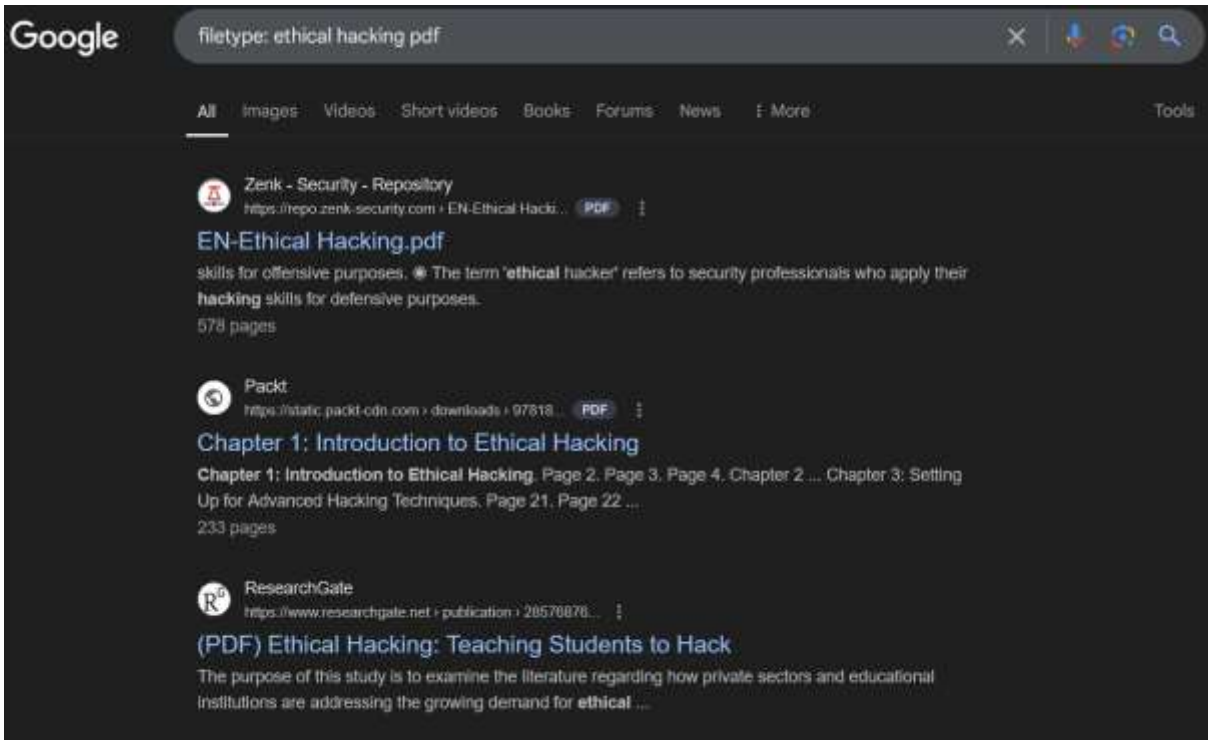
How It Works
- inurl: — tells the search engine to look for pages that have the keyword in their URL.
- <keyword> — the term you want to find in the URL.
- <domain name> — helps narrow results to a specific website (optional but commonly included).

- Example:
  inurl:login site:example.com
- This finds all URLs on example.com that have the word "login" in them (e.g., example.com/user/login, example.com/admin/login, etc.)



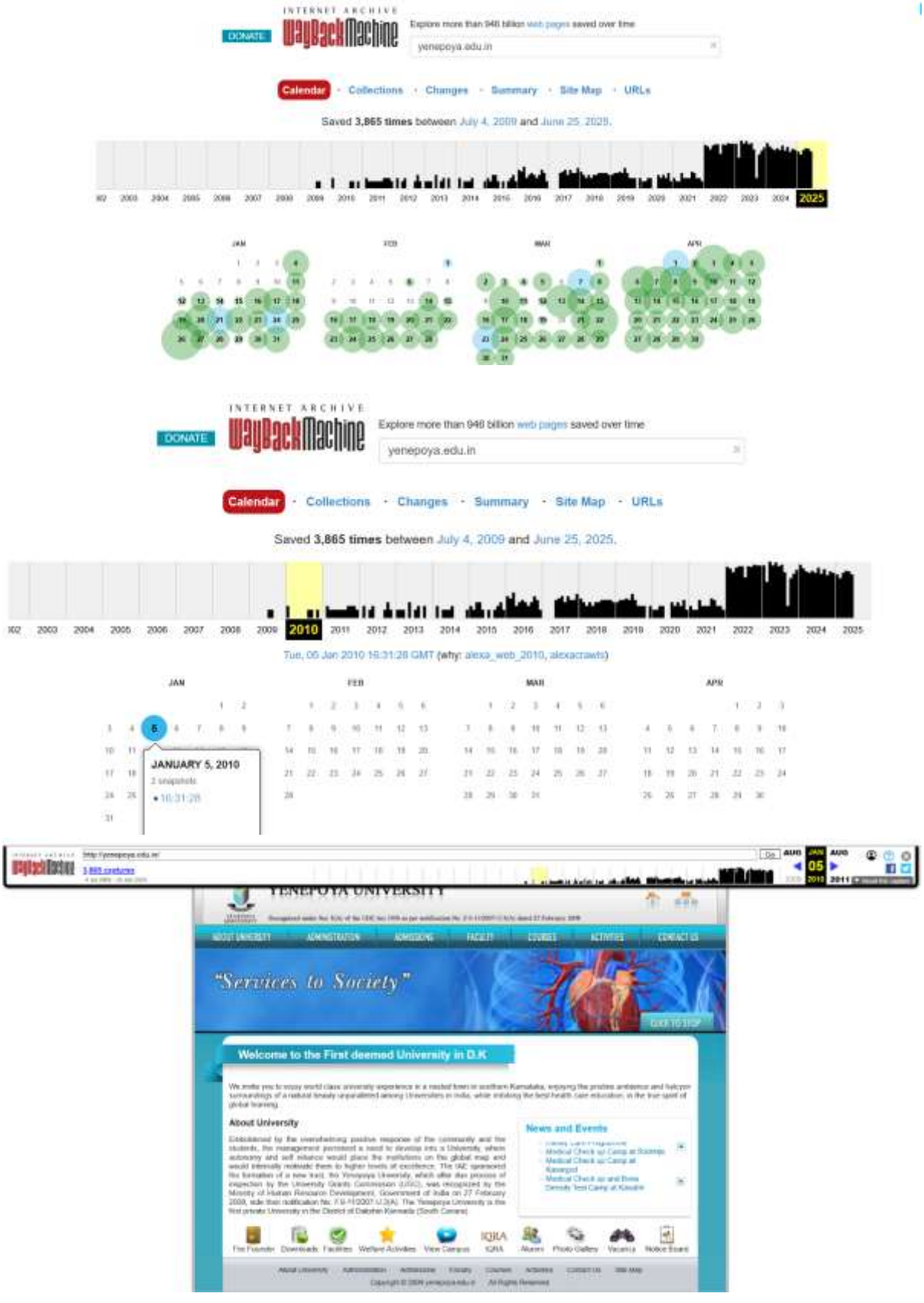
ii. **filetype:<extension> <keyword>**
is used in Google and other search engines to find files of a specific type (like PDF, DOCX, PPT, etc.) that contain the given keyword.
How It Works:

- filetype: — restricts search results to files with a specific extension.
- <extension> — the type of file you're looking for (e.g., pdf, docx, ppt, xls).
- <keyword> — what you're searching for inside the file.



d. archive.org

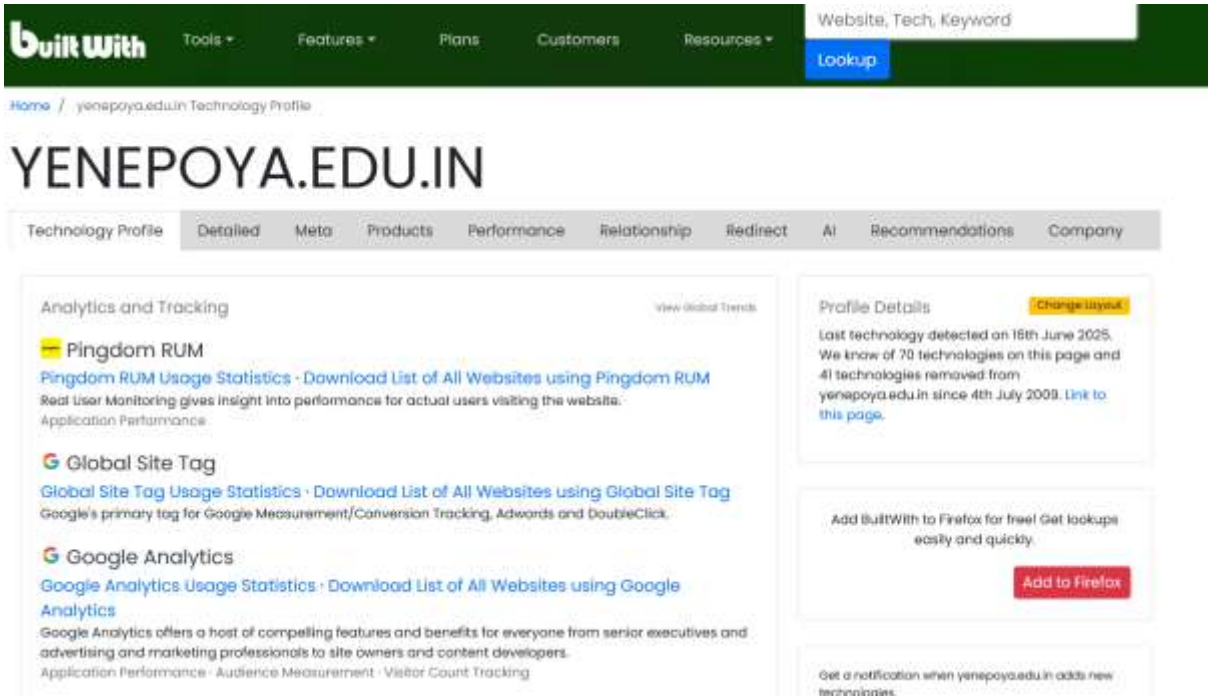
- archive.org, also known as the Internet Archive, is a non-profit digital library that provides free access to historical content on the internet.
- View archived versions of websites from different dates (even if they are deleted now)
- View deleted or changed websites (e.g., research how a site looked in 2005)
- Example:
  - Go to: **https://web.archive.org**
  - Enter: example.com
  - You'll see a calendar with archived snapshots of that website over time.


INTERNET ARCHIVE
WayBackMachine
DONATE
Explore more than 948 billion web pages saved over time
yenepoya.edu.in
Calendar · Collections · Changes · Summary · Site Map · URLs
Saved 3,865 times between July 4, 2009 and June 25, 2025.
JAN
FEB
MAR
APR
Tue, 05 Jan 2010 16:31:28 GMT (why: alexa_web_2010, alexacrawls)
JANUARY 5, 2010
2 snapshots
16:31:28
YENEPOYA UNIVERSITY
ABOUT UNIVERSITY
ADMINISTRATION
ADMISSIONS
FACULTY
COURSES
ACTIVITIES
CONTACT US
"Services to Society"
Welcome to the First deemed University in D.K
About University
News and Events

e. builtwith.com

BuiltWith is a website profiler tool that tells you what technologies a website is using.

Category	Details Provided
Web Server & Hosting	Apache, Nginx, IIS, Cloudflare, etc.
CMS (Content Management System)	WordPress, Joomla, Drupal, Shopify, etc.
E-commerce Platforms	WooCommerce, Magento, BigCommerce
Analytics & Tracking	Google Analytics, Hotjar, Facebook Pixel
Payment Tech	Stripe, PayPal, Razorpay
SSL & Security	HTTPS, SSL issuer, firewall tech like Sucuri, Cloudflare
JavaScript Libraries	React, jQuery, Angular, etc.
Email Providers	Mailchimp, SendGrid, etc.
CDNs & Hosting Providers	Amazon AWS, Akamai, Google Cloud, etc.



f. cve.org

- CVE is a publicly disclosed list of cybersecurity vulnerabilities and exposures, and used worldwide to identify and track security flaws in software and hardware.
- Stay informed about vulnerabilities in libraries/tools you use
- Patch systems based on recent CVEs
- Track newly disclosed vulnerabilities
- Check whether systems are exposed to high-risk CVEs

2. Perform packet sniffing using Wireshark and analyze network traffic to inspect protocols, IP details, port usage, TCP flags, and HTTP methods. Use appropriate filters to extract and interpret specific types of packets.

a. How do you filter and analyze packets transmitted over the HTTP protocol in Wireshark?
http

b. Which display filter is used to show packets involving a specific IP address (either source or destination)?
ip.addr == <IP address>

c. What filter should be applied to capture packets originating from a specific IP address?
ip.src == <Source IP address>

d. How do you filter packets going to a specific destination IP address?
ip.dst == <Destination IP address>

e. What filter can be used to view packets where the SYN flag is set?
tcp.flags.syn == 1

f. How do you identify packets where the TCP connection was reset?
tcp.flags.reset == 1

g. Which filter highlights packets where the TCP connection is finishing (FIN flag set)?
tcp.flags.fin == 1

h. What combination of flags indicates the first step in a TCP handshake?
tcp.flags.syn == 1 && tcp.flags.ack == 0

i. How do you filter packets that use the HTTP methods?
http.request.method == "GET"
http.request.method == "POST"

j. What filter do you use to display traffic on HTTPS port?
tcp.port == 443

k. Which filter is used to view packets on the HTTP port?
tcp.port == 80

l. How can you monitor FTP control traffic using port-based filtering?
tcp.port == 21

3. Demonstrate SQL Injection testing on a target web application using sqlmap. Perform database enumeration, table extraction, and data dumping operations by targeting injectable parameters.

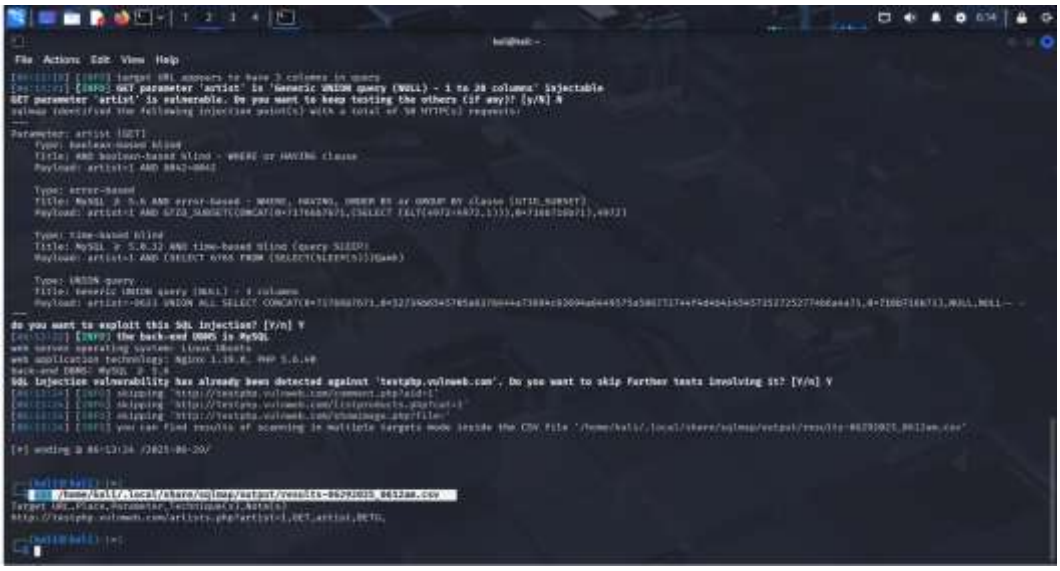
Commands and the Output:

i. sqlmap -u <target URL> --crawl <number> --batch

- This command is used to automatically detect and exploit SQL injection vulnerabilities on a target website.
- -u <target URL> → Specifies the URL of the website you want to test.
- --crawl <number> → Tells sqlmap to crawl the site up to that many levels deep, looking for pages with parameters.
- --batch → Runs the tool automatically without asking the user for confirmation during execution (non-interactive mode).

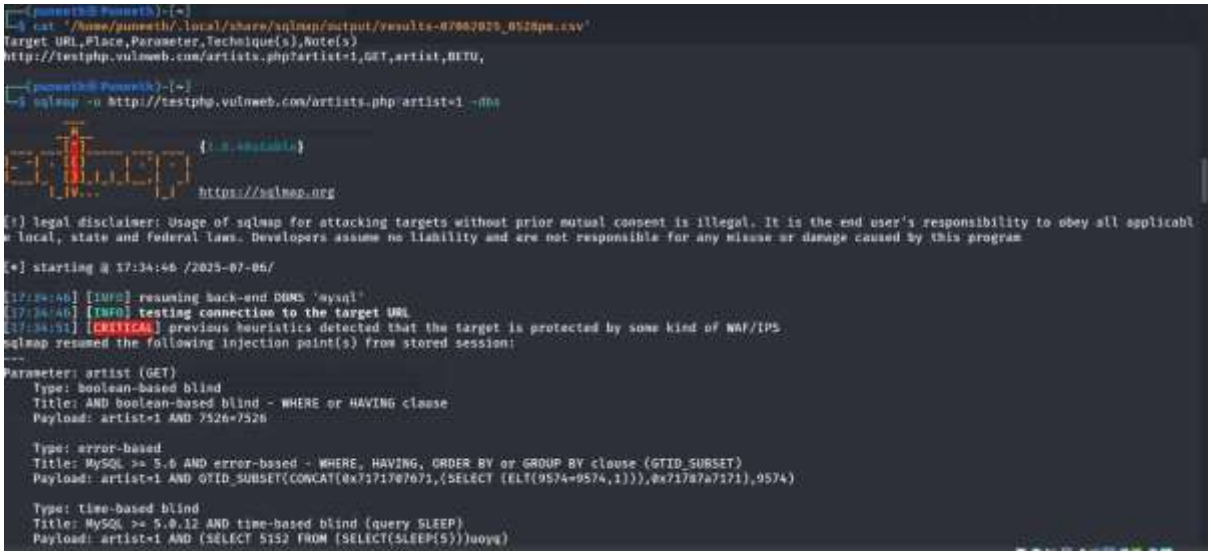
ii. cat <.csv file Path name to get the URL>

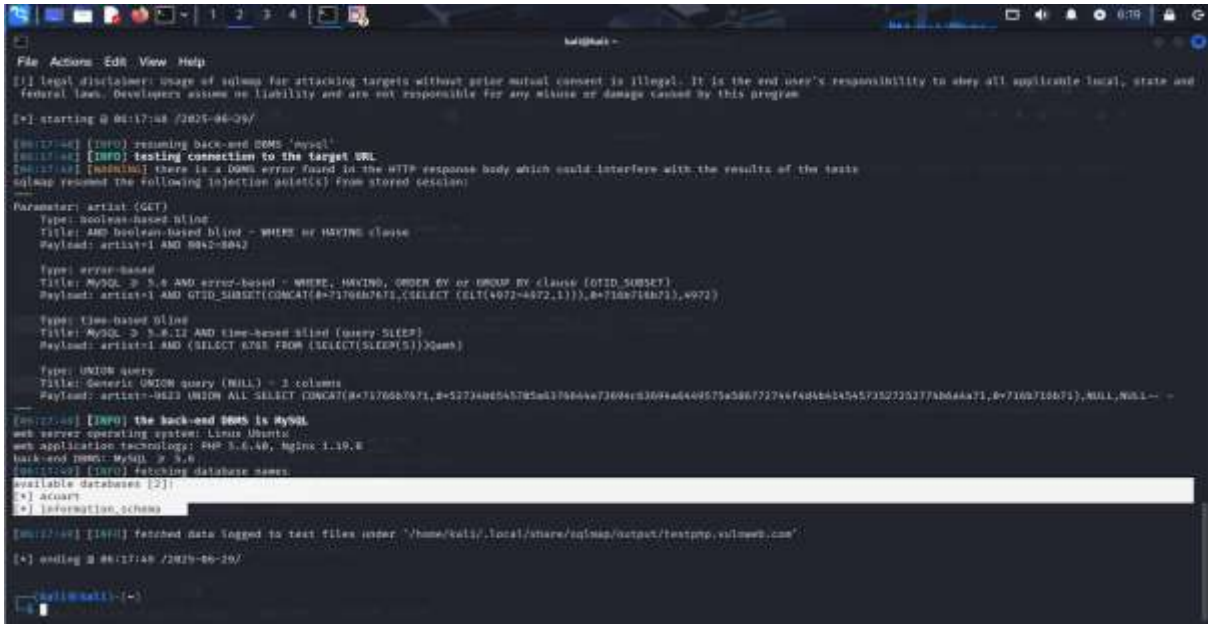
- cat is a Linux/Unix command used to **display the contents** of a file.
- <.csv file path name> should be replaced with the **actual path** to your .csv file that contains URLs.



iii. sqlmap -u <URL from the .csv file> --dbs

- To check if the specified URL is vulnerable to SQL injection and, if so, list all databases in the target system.
- -u <URL> → Specifies the target URL you want to test for SQL injection. Replace <URL> with an actual URL.
- --dbs → Tells sqlmap to enumerate the available databases on the target system if the injection is successful.

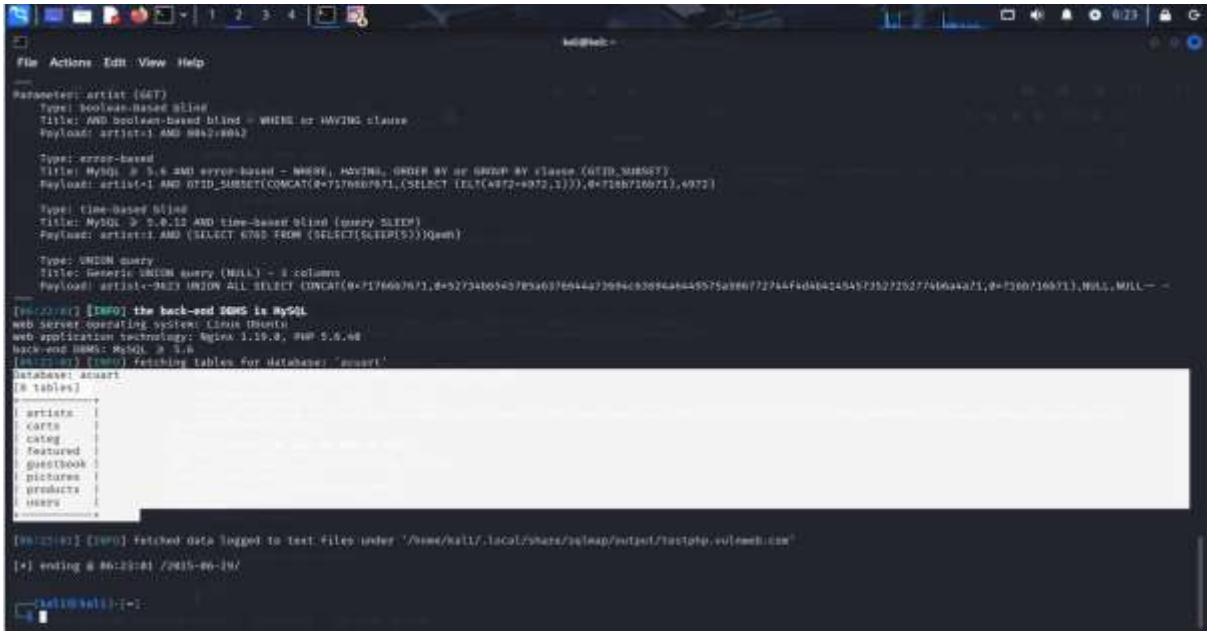




iv. sqlmap -u <URL from the .csv file> -D <database name> --tables

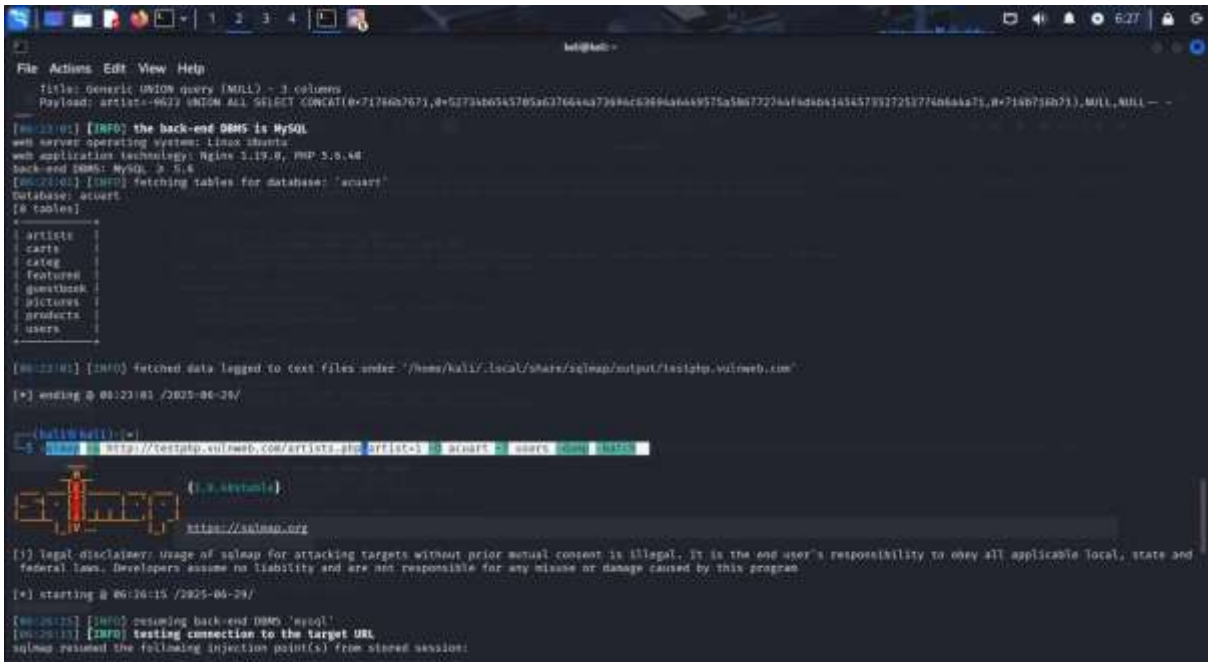
- To list the tables within a specific database from a URL that is vulnerable to SQL injection.
- -u <URL from the .csv file> → Specifies the target URL containing a vulnerable parameter.
- -D <database name> → Specifies the target database (you get this from the previous --dbs command).

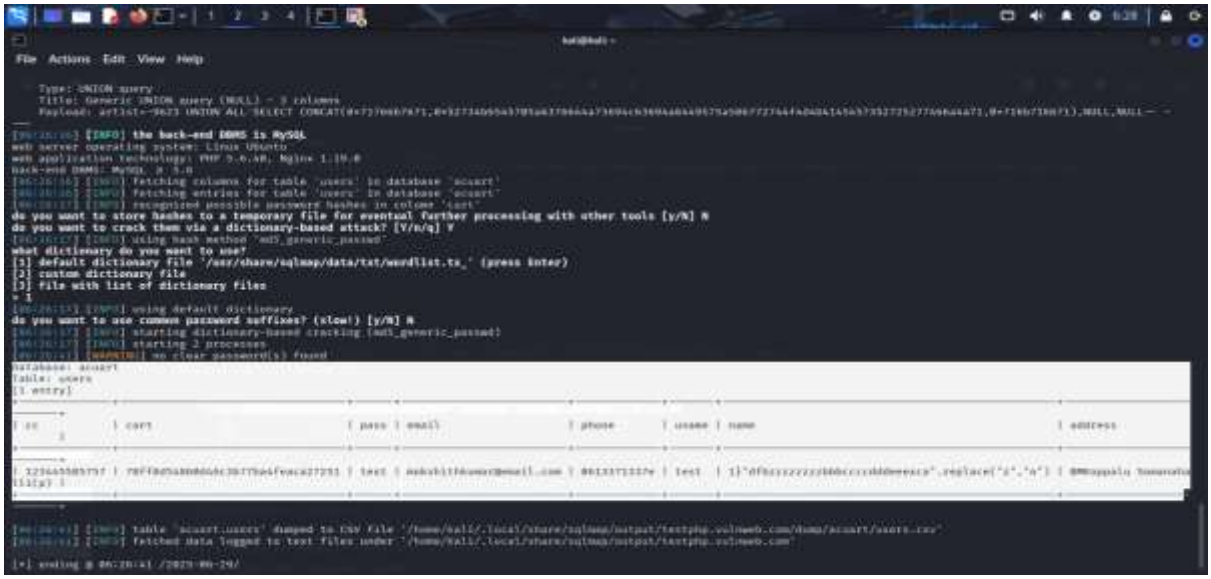
- --tables → Tells sqlmap to list all the tables inside the specified database.



v. sqlmap -u <URL from the .csv file> -D <database name> -T <table name> --dump –batch

- To extract the full data from a specific table in a vulnerable database using SQL injection.
- -T <table name> → Specifies the target table within that database.
- --dump → Tells sqlmap to extract and display all data from the specified table.
- --batch → Runs the command in non-interactive mode, automatically choosing default options (useful for scripts or automation).





vi. cat <.csv file Path name to get the table data in local storage>

- The cat command in Unix/Linux is used to display the contents of a file, including .csv files.
- This is useful to inspect extracted SQL data that you saved into a .csv file.

4. **Perform active network scanning and enumeration on a target system using Nmap. Analyze and interpret the results of different scanning techniques to identify open ports, running services, operating system details, and host availability.**

   i. **Perform a basic TCP scan on the target host to identify open ports and running services.**

   nmap <target>

   - (with <target> replaced by an IP address or domain) runs a basic Nmap scan. It performs a TCP connect scan (default scan type) to check which ports are open on the target host.
   - When you run:

     nmap scanme.nmap.org
   - You can expect an output like this:

```
bash

Starting Nmap 7.93 ( https://nmap.org ) at 2025-07-14 10:22 IST
Nmap scan report for scanme.nmap.org (45.33.32.156)
Host is up (0.10s latency).
Not shown: 997 closed ports
PORT     STATE SERVICE
22/tcp   open  ssh
80/tcp   open  http
9929/tcp open  nping-echo

Nmap done: 1 IP address (1 host up) scanned in 3.52 seconds
```

   - What This Output Tells You

Section	Details
Target Info	IP address and domain name of the scanned host
Host Status	Whether the host is up (reachable)
Port Info	Shows each open port, its status, and the service running on that port
Summary	Total ports scanned, hosts up, and time taken

   - Use Case
     - Check if a server is online
     - Discover open ports (e.g., SSH on port 22, HTTP on port 80)
     - Find exposed services that could be potential attack surfaces

   ii. **Conduct a stealth (SYN) scan on the target and observe how it differs from a basic TCP scan.**

   nmap -sS <target>

   - Performs a TCP SYN scan on the specified target (replace <target> with a domain or IP). This is also called a half-open scan.
     - When you run:

       nmap -sS scanme.nmap.org

- You can expect output similar to:

```
Starting Nmap 7.93 ( https://nmap.org ) at 2025-07-14 10:30 IST
Nmap scan report for scanme.nmap.org (45.33.32.156)
Host is up (0.12s latency).
Not shown: 996 closed ports
PORT     STATE SERVICE
22/tcp   open  ssh
80/tcp   open  http
9929/tcp open  nping-echo
```

- What the Output Means

Part	Explanation
22/tcp open ssh	Port 22 is open and running SSH
80/tcp open http	Port 80 is open and running HTTP (a web server)
Host is up	The server is online and responding
Not shown: …	Many other ports are closed and not shown to simplify output

- Why -sS (SYN) Scan?
  - **Stealthy:** Sends SYN packet and analyzes the response without completing the full TCP handshake.
  - **Faster:** Scans quicker than a full TCP connect scan.
  - **Less detectable:** Often bypasses basic firewalls and logging systems.

iii. Perform a TCP connect scan to identify port connectivity using the full TCP handshake.

nmap -sT <target>

- Performs a TCP Connect Scan, which is Nmap's default scan type if you're not running as root/admin.
- When you run:
  - **nmap -sT example.com**
- You'll get an output like:

```
Starting Nmap 7.93 ( https://nmap.org ) at 2025-07-14
Nmap scan report for example.com (93.184.216.34)
Host is up (0.021s latency).
Not shown: 995 closed ports
PORT     STATE SERVICE
22/tcp   open  ssh
80/tcp   open  http
443/tcp  open  https
```

- What It Tells You
  - **Open ports:** Lists TCP ports that are open (e.g., 80 for HTTP, 22 for SSH).
  - **State:** Shows whether each port is open, closed, or filtered.
  - **Service:** Attempts to identify the common service running on that port.
- What Happens Internally
  - Nmap completes the TCP 3-way handshake with each target port.
  - It's easier to detect by intrusion detection systems (IDS) and firewalls.
  - Requires no special privileges (can be run as a normal user).
- Use Case
  - When you don't have root/admin rights on your machine.
  - When you need a safe, complete connection to check reachable ports.

iv. Execute a UDP scan to identify open or accessible UDP services on the target system.

nmap -sU <target>

- Performs a UDP scan on the specified target.
- Expected Output

  nmap -sU example.com
- You may get output like:

```
Starting Nmap 7.93 ( https://nmap.org ) at 2025-07-14
Nmap scan report for example.com (93.184.216.34)
Host is up (0.080s latency).
Not shown: 998 closed ports
PORT     STATE         SERVICE
53/udp   open          domain
123/udp  open          ntp
```

- What It Tells You
  - Lists open/closed/filtered UDP ports.
  - Identifies services running on open UDP ports (e.g., DNS on 53, NTP on 123).
  - Since UDP is connectionless, it may show ports as:
    - **open:** the port is definitely open
    - **open|filtered:** Nmap can't be sure due to no response
    - **filtered:** response blocked by a firewall
- What Happens Internally

- Sends UDP packets to each port.
- Determines state based on response or lack thereof.
- No 3-way handshake like TCP, making it slower and harder to confirm.

- Use Case:
  - Used to detect services that run over UDP like:
    DNS (53), SNMP (161), NTP (123), TFTP (69)
  - Helpful for comprehensive network security assessments.
- Note:
  - UDP scans can be much slower than TCP scans and less reliable, since many ports just don't respond at all.

v. Run an aggressive scan to gather comprehensive system information, including OS, services, and scripts.

nmap -A <target>

- Performs an aggressive scan on the given target.
- What It Does (Aggressive Scan):
  - -A enables multiple advanced features at once:
    - OS detection
    - Version detection of services
    - Script scanning (default NSE scripts)
    - Traceroute
- Example Output
  nmap -A example.com
- You may get output like:

```
PORT     STATE SERVICE VERSION
22/tcp   open  ssh     OpenSSH 7.6 (protocol 2.0)
80/tcp   open  http    Apache httpd 2.4.29
...
OS details: Linux 4.15 - 5.4
Network Distance: 8 hops
```

- What You Can Expect
  - Open ports
  - Detected services and versions
  - Operating system and device type (if detectable)
  - Traceroute output showing network path
  - Possible vulnerabilities if scripts are triggered
- Use With Caution
  - Loud on the network: easily detected by intrusion detection systems (IDS).

- May be blocked or rate-limited by firewalls or defensive systems.
- Use on authorized systems only, especially in organizational or public environments.

- Tip:
  - Use nmap -A -T4 <target> for a faster aggressive scan with less timeout delays.

vi. Detect the operating system running on the target host using Nmap's OS detection.

nmap -O <target>

- Perform Operating System (OS) detection on the specified target.
- What It Does:
  - -O enables OS fingerprinting, which tries to identify the target system's operating system based on how it responds to various network probes.
- Example

  nmap -O example.com
- Expected Output

```
PORT    STATE  SERVICE
22/tcp  open   ssh
80/tcp  open   http
Device type: general purpose
Running: Linux 4.X|5.X
OS CPE: cpe:/o:linux:linux_kernel:4
OS details: Linux 4.8 - 5.4
```

- What You Can Expect:
  - OS name and version (e.g., Linux 5.x, Windows 10, etc.)
  - Device type (e.g., router, general-purpose computer)
  - Confidence level of the match
  - CPE (Common Platform Enumeration) info for further vulnerability analysis.
- Notes:
  - Requires root/administrator privileges on most systems for accurate results.
  - May fail or be inaccurate if the target uses a firewall, IDS/IPS, or obscures TCP/IP stack fingerprints.
  - Works better with open ports and predictable responses.
- Use Case:
  - Very useful for penetration testers, network admins, or security researchers trying to understand the environment or plan further assessments.

vii. Identify versions of services running on the open ports of the target system.

nmap -sV <target>

- Perform service/version detection on the specified target.
- The -sV flag enables Nmap to:
  - Probe open ports
  - Determine what service is running on them (e.g., Apache, SSH, MySQL)
  - Attempt to determine the version of that service (e.g., Apache 2.4.52)
- Example:

  nmap -sV example.com
- Expected Output

```
PORT     STATE SERVICE VERSION
22/tcp   open  ssh     OpenSSH 7.6p1 Ubuntu 4ubuntu0.3
80/tcp   open  http    Apache httpd 2.4.29
443/tcp  open  ssl/http Apache httpd 2.4.29
```

- What You Can Expect
  - Open ports
  - Service names (e.g., SSH, HTTP)
  - Software version (e.g., OpenSSH 7.6p1)
  - Sometimes additional info like OS or SSL certificate details
- Notes:
  - Slower than a basic scan because it sends additional probes for version detection.
  - Can be used along with -p to limit to specific ports.
  - Useful for vulnerability scanning, since known versions can be matched with known exploits.
- Use Case:
  - Ideal for penetration testers and system administrators to detect outdated or vulnerable software versions running on a networked system.

viii. Scan a specific range of ports (1 to 1000) to check which services are active in that range.

nmap -p 1-1000 <target>

- This tells Nmap to scan ports 1 through 1000 on the specified target (which can be a domain name, IP address, or hostname).
- Expected Output:
  - Nmap will list:
    - The ports in that range (1–1000) that are open, closed, or filtered.

        - The services (like SSH, HTTP) running on the open ports (if detectable)
    - Example Output:

    - Purpose:
        - This is useful when:
            - You want to limit the scan to commonly used ports (instead of scanning all 65535 ports)
            - You need a quick scan of likely service ports
    - Tip:
        - You can customize the port range (e.g., -p 20-30) or list specific ports (-p 21,22,80) for faster and more targeted scanning.

ix. Perform a faster scan using the T4 timing template and analyze the effect on scanning speed and accuracy.

nmap -T4 <target>

- What it does:
    - The -T4 option in Nmap sets the timing template to level 4 (Aggressive), which speeds up the scan.
- What to expect in the output:
    - You will receive the same results as a regular scan, but faster, depending on network conditions. It may still include:
        - Open, closed, or filtered ports
        - Detected services
        - Scan duration time
- Timing Levels in Nmap (-T):

Timing Level	Description	Use Case
-T0	Paranoid (very slow)	IDS evasion
-T1	Sneaky	IDS evasion
-T2	Polite	Low-priority scan (less bandwidth)
-T3	Normal (default)	General purpose
-T4	Aggressive	Fast scans on reliable networks
-T5	Insane (very fast)	Very fast scan (risky, may be inaccurate)

- Note:
    - -T4 is recommended for fast scans on local or stable networks.
    - May cause network congestion or trigger security alerts on target systems.

x. **Scan a host that blocks ping requests by disabling host discovery.**
nmap -Pn <target>

- What it does:
  - The -Pn option disables host discovery (ping scan). This means Nmap will assume the target is online, even if it doesn't respond to ping (ICMP) requests.
- What you can expect in the output:
  - A port scan results even if the host doesn't respond to ICMP or ping probes.
  - Lists of open, closed, or filtered ports.
  - Service detection, if combined with flags like -sV.
- When to use it:
  - When scanning firewalled systems that block ICMP (ping).
  - When scanning servers that appear down but might just ignore ping requests.
  - When scanning cloud-based services (e.g., AWS, Azure) that often block ping.
- Caution:
  - May waste time trying to scan offline hosts.
  - Always combine with specific scan types like -sS, -sV, etc., for meaningful results.
- Example:
  nmap -Pn -sS -p 80,443 example.com
  - This scans ports 80 and 443 on example.com even if it doesn't reply to pings.

xi. **Conduct a ping scan to discover active hosts in a network without performing a port scan.**
nmap -sn <target>

- What it does:
  - Performs a ping scan only (also called a "host discovery" scan).
  - It does not scan ports.
  - Checks which hosts are up (online) on a given network.
- What you can expect in the output:
  - A list of live/active hosts (devices that respond to ping or ARP).
  - Their IP addresses and possibly hostnames (if DNS resolution is enabled).
  - No information about open ports or services.
- When to use it:
  - To discover live devices on a network quickly.
  - As a pre-scan before running a full port scan.

- Useful in large networks to check online hosts.

- Example:

  nmap -sn 192.168.1.0/24

  - This will ping all IPs in the range 192.168.1.0 to 192.168.1.255 and report which are up.

xii. Perform a verbose scan to observe detailed scanning progress and output.

nmap -v <target>

- Explanation:
  - nmap → Runs the Nmap network scanning tool.
  - -v → Enables verbose mode, giving more detailed output as Nmap performs the scan.
  - <target> → This can be an IP address (e.g., 192.168.1.1) or a domain name (e.g., scanme.nmap.org).
- Expected Output:
  - Verbose output will show:
    - Starting scan message
    - DNS resolution (if domain is given)
    - Ping attempts
    - Scanning progress (like port status updates)
    - Final results with open ports and services
- Command (Input):

  nmap -v scanme.nmap.org

- Output (Verbose):

```
Starting Nmap 7.93 ( https://nmap.org ) at 2025-07-15 13:05 IST
Initiating Ping Scan at 13:05
Scanning scanme.nmap.org (45.33.32.156) [4 ports]
Completed Ping Scan at 13:05, 0.22s elapsed (1 total hosts)
Initiating Parallel DNS resolution of 1 host. at 13:05
Completed Parallel DNS resolution of 1 host. at 13:05, 0.04s elapsed
Initiating SYN Stealth Scan at 13:05
Scanning scanme.nmap.org (45.33.32.156) [1000 ports]
Discovered open port 22/tcp on 45.33.32.156
Discovered open port 80/tcp on 45.33.32.156
Completed SYN Stealth Scan at 13:05, 3.12s elapsed (1000 total ports)
Nmap scan report for scanme.nmap.org (45.33.32.156)
Host is up (0.18s latency).
Other addresses for scanme.nmap.org (not scanned): 2600:3c01::f03c:91ff:fe18:bb2f

PORT     STATE SERVICE
22/tcp   open  ssh
80/tcp   open  http

Read data files from: /usr/bin/../share/nmap
Nmap done: 1 IP address (1 host up) scanned in 3.42 seconds
```

- What You Learned from This Scan:
  - Target is online
  - Open ports found: 22/tcp (SSH) and 80/tcp (HTTP)
  - You get real-time progress because of -v

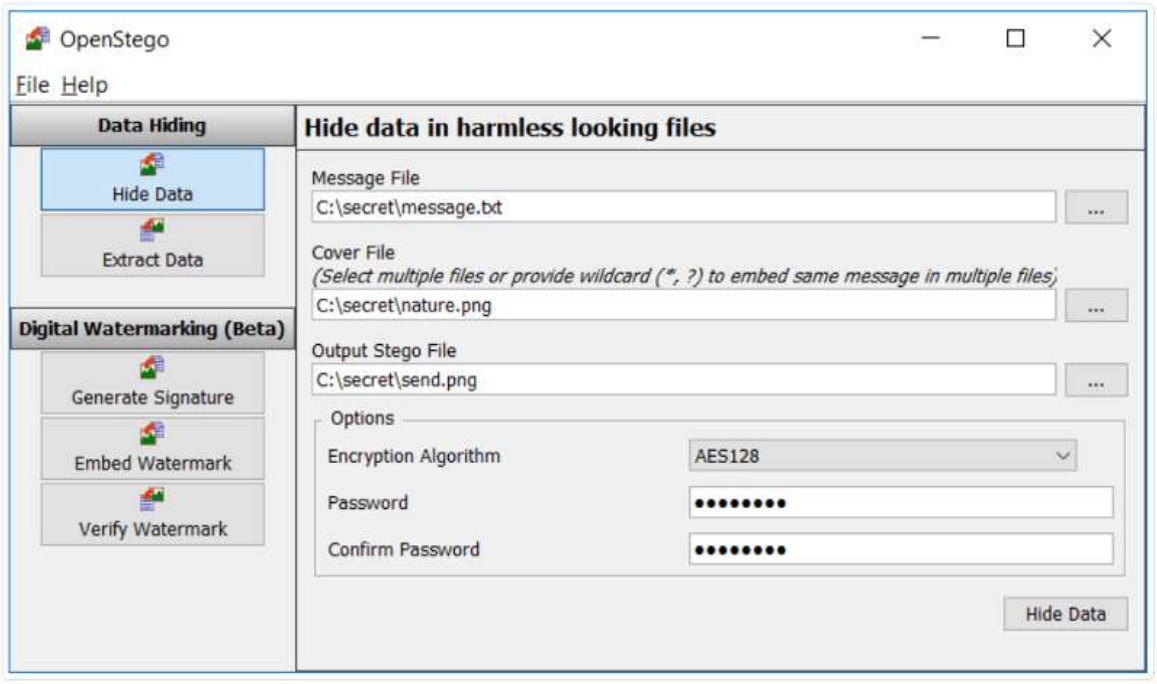
5. Perform steganography by hiding a secret text message inside an image and then extract it successfully.

Steps

Hiding Data:

a. Open the OpenStego application.
b. Select "Hide Data" mode
c. Choose Message File: select the file you wish to hide
d. Choose Cover File: select the image file that will serve as the container for your hidden data.
e. Specify Output Stego File: Navigate to the desired output directory, and provide a name for the new image file that will contain the hidden data.
f. Set Password
g. Hide Data

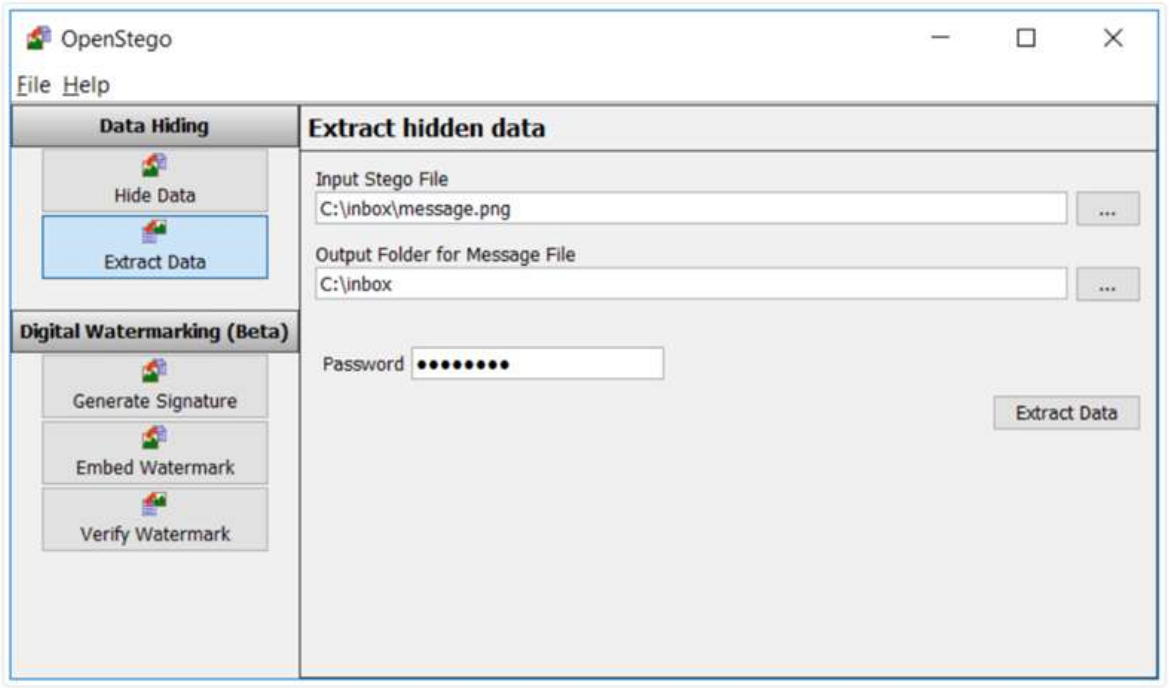
Hide data



Extracting Data:

a. Select "Extract Data" mode.
b. Under Stego File, select the image that contains the hidden data (e.g., output_stego.jpg).
c. Specify the Output Folder where the extracted file will be saved.
d. Enter the same password used during hiding.
e. Extract Data

Extract data



PART – B

1. **Develop a Python program to demonstrate multiple encoding and decoding schemes commonly used in web applications. Your program should implement Base64, URL encoding, HTML encoding, and Hex encoding for both encoding and decoding functionalities.**

Code:

```
import base64
import urllib.parse
import html

def encode_decode_base64(text):
    encoded = base64.b64encode(text.encode()).decode()
    decoded = base64.b64decode(encoded.encode()).decode()
    return encoded, decoded

def encode_decode_url(text):
    encoded = urllib.parse.quote(text)
    decoded = urllib.parse.unquote(encoded)
    return encoded, decoded

def encode_decode_html(text):
    encoded = html.escape(text)
    decoded = html.unescape(encoded)
    return encoded, decoded

def encode_decode_hex(text):
    encoded = text.encode("utf-8").hex()
    decoded = bytes.fromhex(encoded).decode("utf-8")
    return encoded, decoded

input_text = "<script>alert('XSS')</script>"

print("Original Text:", input_text)

b64_enc, b64_dec = encode_decode_base64(input_text)
print("\nBase64 Encoding:", b64_enc)
print("Base64 Decoding:", b64_dec)

url_enc, url_dec = encode_decode_url(input_text)
print("\nURL Encoding:", url_enc)
print("URL Decoding:", url_dec)
```

```
html_enc, html_dec = encode_decode_html(input_text)
print("\nHTML Encoding:", html_enc)
print("HTML Decoding:", html_dec)

hex_enc, hex_dec = encode_decode_hex(input_text)
print("\nHex Encoding:", hex_enc)
print("Hex Decoding:", hex_dec)
```

Output:

2. Develop a basic Python-based HTTP server and client to simulate and analyze HTTP GET and POST requests. Observe the structure of HTTP headers and demonstrate how modifying headers such as User-Agent and Referer affects server response.

To Execute the Server:

i. Save the server code in a file, e.g.: http_server.py
ii. Open terminal and run: python3 http_server.py
iii. Observe Server: Server running on http://localhost:8080/

To Execute the Client:

i. Save it in a file, e.g.: http_client.py
ii. In a new terminal window, run: python3 http_client.py (or split terminal)
iii. Observe both:
   - The terminal where the server is running (it shows headers and body)
   - The terminal where you ran the client (it shows responses)

Code: http_server.py

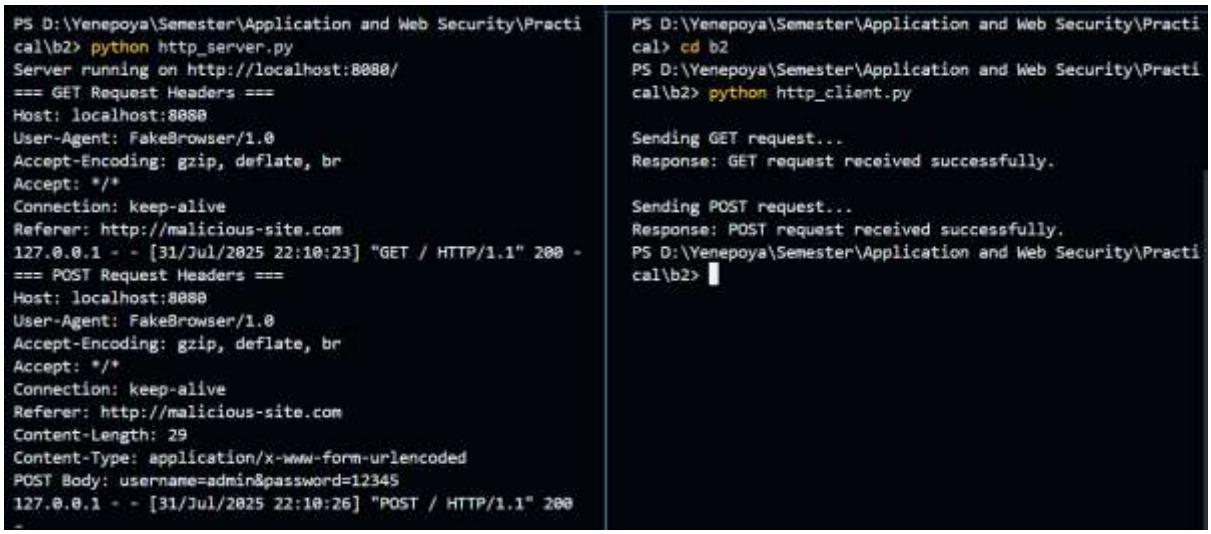
```
from http.server import BaseHTTPRequestHandler, HTTPServer

class SimpleHTTPRequestHandler(BaseHTTPRequestHandler):

    def do_GET(self):
        print("=== GET Request Headers ===")
        for header, value in self.headers.items():
            print(f"{header}: {value}")

        self.send_response(200)
        self.end_headers()
        self.wfile.write(b"GET request received successfully.")

    def do_POST(self):
        print("=== POST Request Headers ===")
        for header, value in self.headers.items():
            print(f"{header}: {value}")

        content_length = int(self.headers.get('Content-Length', 0))
        post_data = self.rfile.read(content_length)
        print(f"POST Body: {post_data.decode()}")

        self.send_response(200)
        self.end_headers()
        self.wfile.write(b"POST request received successfully.")
```

```
def run_server():
    server_address = ('localhost', 8080)
    httpd = HTTPServer(server_address, SimpleHTTPRequestHandler)
    print("Server running on http://localhost:8080/")
    httpd.serve_forever()

if __name__ == '__main__':
    run_server()
```

Code: http_client.py

```
import requests

url = 'http://localhost:8080'

custom_headers = {
'User-Agent': 'FakeBrowser/1.0',
'Referer': 'http://malicious-site.com'
}

print("\nSending GET request...")
response = requests.get(url, headers=custom_headers)
print("Response:", response.text)

print("\nSending POST request...")
post_data = {'username': 'admin', 'password': '12345'}
response = requests.post(url, headers=custom_headers, data=post_data)
print("Response:", response.text)
```

Output:



3. Develop a simple Python web server using Flask or http.server that demonstrates different HTTP response codes (e.g., 200, 404, 403, 500). Analyze the behavior of both the server and the client when these status codes are returned.

Code: server.py

```
from flask import Flask

app = Flask(__name__)

@app.route("/success")
def success():
    return "200 OK - Success", 200

@app.route("/forbidden")
def forbidden():
    return "403 Forbidden - Access Denied", 403

@app.route("/notfound")
def not_found():
    return "404 Not Found - Page doesn't exist", 404

@app.route("/error")
def server_error():
    return "500 Internal Server Error - Something went wrong", 500

if __name__ == "__main__":
    app.run(debug=True)
```

Code: client.py

```
import requests

base_url = "http://127.0.0.1:5000"
endpoints = ["/success", "/forbidden", "/notfound", "/error"]

for endpoint in endpoints:
    url = base_url + endpoint
    response = requests.get(url)
    print(f"\nRequest to {url}")
    print(f"Status Code: {response.status_code}")
    print(f"Response Body: {response.text}")
```

Output:


PS D:\Yenepoya\Semester\Application and Web Security\Practical\b3> python server.py
* Serving Flask app 'server' (lazy loading)
* Environment: production
WARNING: This is a development server. Do not use it in a production deployment.
Use a production WSGI server instead.
* Debug mode: on
* Restarting with watchdog (windowsapi)
* Debugger is active!
* Debugger PIN: 108-349-042
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
127.0.0.1 - - [04/Aug/2025 14:34:07] "GET /success HTTP/1.1" 200 -
127.0.0.1 - - [04/Aug/2025 14:34:07] "GET /forbidden HTTP/1.1" 403 -
127.0.0.1 - - [04/Aug/2025 14:34:07] "GET /notfound HTTP/1.1" 404 -
127.0.0.1 - - [04/Aug/2025 14:34:07] "GET /error HTTP/1.1" 500 -
PS D:\Yenepoya\Semester\Application and Web Security\Practical\b3> python client.py
Request to http://127.0.0.1:5000/success
Status Code: 200
Response Body: 200 OK - Success
Request to http://127.0.0.1:5000/forbidden
Status Code: 403
Response Body: 403 Forbidden - Access Denied
Request to http://127.0.0.1:5000/notfound
Status Code: 404
Response Body: 404 Not Found - Page doesn't exist
Request to http://127.0.0.1:5000/error
Status Code: 500
Response Body: 500 Internal Server Error - Something went wrong
PS D:\Yenepoya\Semester\Application and Web Security\Practical\b3>

4. Simulate a brute-force attack in a secure, offline environment by building a basic Python Flask login system with hardcoded user credentials. Then, implement a separate Python script to perform login attempts using a wordlist. Analyze and demonstrate how the system responds to repeated failed login attempts.

To Execute:

- In Terminal Tab 1: Run the Flask server – python login_server.py
  - Observe: * Running on http://127.0.0.1:5000/
- In Terminal Tab 2: Run the brute-force script – python brute_force.py

Code: login_server.py

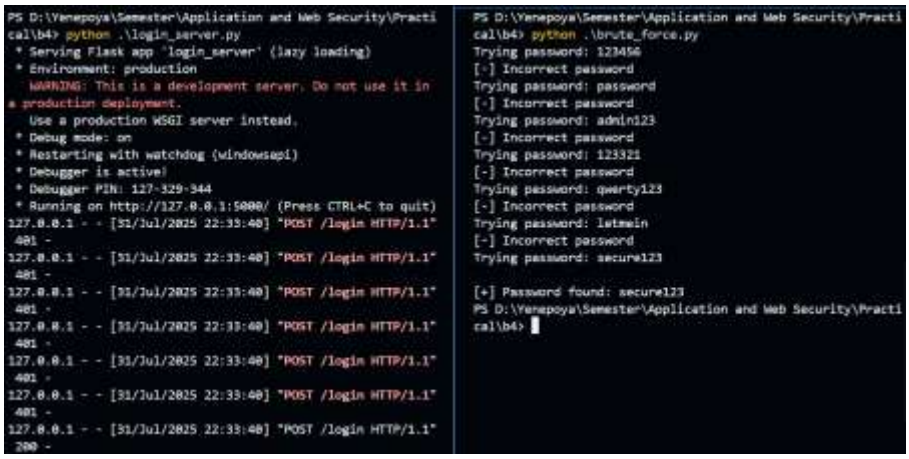
```
from flask import Flask, request, jsonify

app = Flask(__name__)

VALID_USERNAME = 'admin'
VALID_PASSWORD = 'secure123'

@app.route('/login', methods=['POST'])
def login():
    data = request.get_json()

    username = data.get('username')
    password = data.get('password')
    if username == VALID_USERNAME and password == VALID_PASSWORD:
        return jsonify({'status': 'success', 'message': 'Login successful'}), 200
    else:
        return jsonify({'status': 'fail', 'message': 'Invalid credentials'}), 401

if __name__ == '__main__':
    app.run(debug=True, port=5000)
```

Code: brute_force.py

```
import requests

url = 'http://127.0.0.1:5000/login'

username = 'admin'

with open('wordlist.txt', 'r') as file:
    for line in file:
        password = line.strip()  # Remove newline and spaces
```

```
        print(f'Trying password: {password}')

        response = requests.post(url, json={'username': username,
'password': password})

        if response.status_code == 200:
            print(f'\n[+] Password found: {password}')
            break
        else:
            print('[-] Incorrect password')
```

Output:



5. Write a Python program to implement the Caesar Cipher that supports encryption, decryption, and brute-force decryption through a menu-driven interface.

Code:

```
def encrypt(text, key):
    result = ""
    for char in text:
        if char.isalpha():  # only encrypt letters
            shift = 65 if char.isupper() else 97
            result += chr((ord(char) - shift + key) % 26 + shift)
        else:
            result += char
    return result


def decrypt(text, key):
    result = ""
    for char in text:
        if char.isalpha():
            shift = 65 if char.isupper() else 97
            result += chr((ord(char) - shift - key) % 26 + shift)
        else:
            result += char
    return result


def brute_force(text):
    print("\n--- Brute Force Results ---")
    for k in range(1, 26):  # all possible Caesar cipher keys
        print(f"Key {k}: {decrypt(text, k)}")


# Main program loop
while True:
    print("\nEncryption-Decryption Program")
    print("1. Encrypt")
    print("2. Decrypt")
    print("3. Brute Force Decrypt (try all keys)")
    print("4. Exit")

    choice = int(input("Enter your choice: "))

    if choice == 1:
```

```
        plain_text = input("Enter the plain text: ")
        key = int(input("Enter the key value: "))
        cipher_text = encrypt(plain_text, key)
        print("Cipher Text:", cipher_text)

    elif choice == 2:
        cipher_text = input("Enter the cipher text: ")
        key = int(input("Enter the key value: "))
        plain_text = decrypt(cipher_text, key)
        print("Decrypted Text:", plain_text)

    elif choice == 3:
        cipher_text = input("Enter the cipher text: ")
        brute_force(cipher_text)

    elif choice == 4:
        print("Exiting program...")
        break

    else:
        print("Invalid choice! Try again.")
```

Output:

```
PS C:\Users\Dell\Downloads> python .\Shift_cipherse4.py

Encryption-Decryption Program
1. Encrypt
2. Decrypt
3. Brute Force Decrypt (try all keys)
4. Exit
Enter your choice: 1
Enter the plain text: Hello World
Enter the key value: 12
Cipher Text: Tqxxa Iadxp

Encryption-Decryption Program
1. Encrypt
2. Decrypt
3. Brute Force Decrypt (try all keys)
4. Exit
Enter your choice:
```

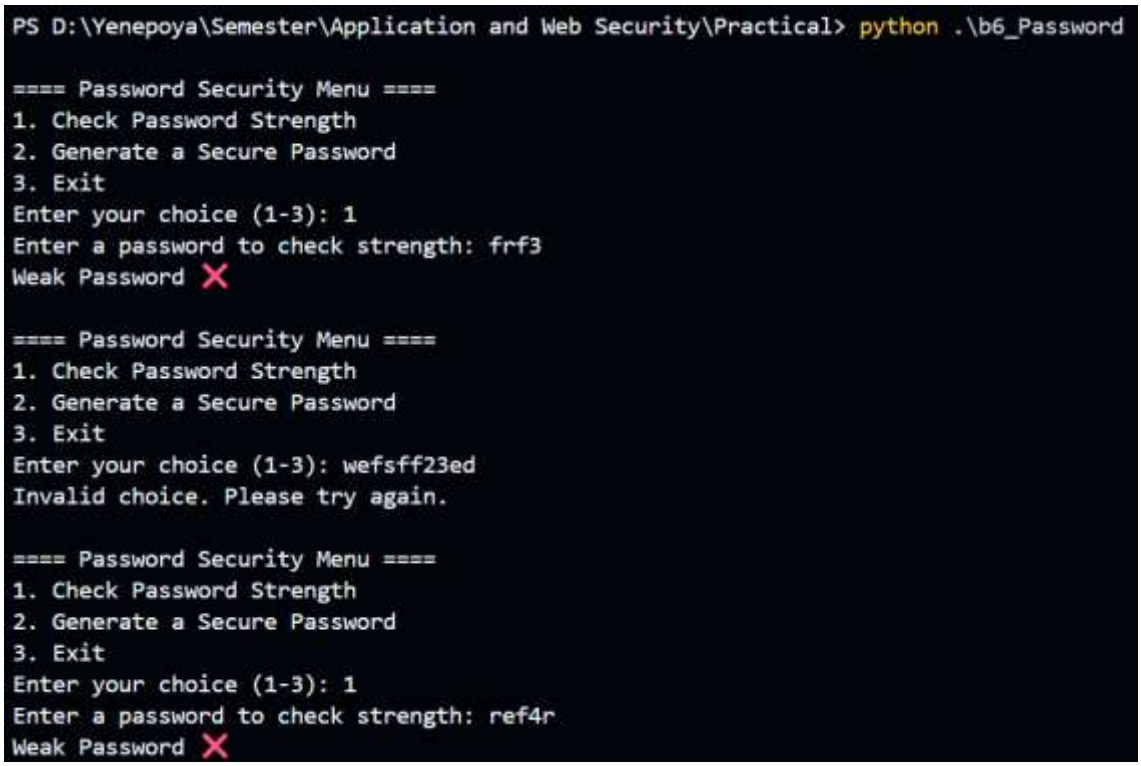
6. Develop a menu-driven Python program to demonstrate password security management. The program should allow users to (i) generate a random secure password based on a specified length, and (ii) validate the strength of a given password using rules such as length, use of uppercase and lowercase letters, digits, and special characters.

Code:

```
import re
import random
import string

# Function to generate a secure random password
def create_password():
    length = int(input("Enter password length: "))
    chars = string.ascii_letters + string.digits + string.punctuation
    password = "".join(random.choice(chars) for _ in range(length))
    print("Generated Password:", password)

# Function to check the strength of a password
def check_strength():
    password = input("Enter a password to check strength: ")
    strength = 0

    # Rules for password strength
    if len(password) >= 8:
        strength += 1
    if re.search("[a-z]", password):
        strength += 1
    if re.search("[A-Z]", password):
        strength += 1
    if re.search("[0-9]", password):
        strength += 1
    if re.search("[@#$%^&*()_+=!]", password):
        strength += 1

    # Strength classification
    if strength == 5:
        print("Strong Password 💪")
    elif strength >= 3:
        print("Medium Strength Password ⚡")
    else:
        print("Weak Password ❌")
```

```
# Main menu loop
def main():
    while True:
        print("\n==== Password Security Menu ====")
        print("1. Check Password Strength")
        print("2. Generate a Secure Password")
        print("3. Exit")

        choice = input("Enter your choice (1-3): ")

        if choice == "1":
            check_strength()
        elif choice == "2":
            create_password()
        elif choice == "3":
            print("Exiting... Goodbye!")
            break
        else:
            print("Invalid choice. Please try again.")

# Run the program
if __name__ == "__main__":
    main()
```

Output:



7. Simulate the use of cookies and server-side sessions to manage user authentication and session state in a Flask-based Python web application. Demonstrate how session tampering is prevented by securely storing session data on the server.

Code:

```
from flask import Flask, request, session, redirect, url_for, render_template_string, abort, make_response
from functools import wraps

app = Flask(__name__)
app.secret_key = 'secureSecretKey123456'

# Simulated user database with roles
users = {
    'admin': {'password': 'admin123', 'role': 'admin'},
    'user1': {'password': 'user123', 'role': 'user'}
}

# Login page template
login_template = '''
<h2>Login</h2>
<form method="post">
  Username: <input type="text" name="username"><br>
  Password: <input type="password" name="password"><br>
  <input type="submit" value="Login">
</form>
'''

# Dashboard page
dashboard_template = '''
<h2>Welcome, {{ user }}</h2>
<p>Role: {{ role }}</p>
<p><a href="/admin">Go to Admin Panel</a></p>
<p><a href="/logout">Logout</a></p>
'''

# Admin page
admin_template = '''
<h2>Admin Panel</h2>
<p>Only for Admins!</p>
<p><a href="/dashboard">Back to Dashboard</a></p>
'''
```

```
# Role-checking decorator
def role_required(required_role):
    def wrapper(f):
        @wraps(f)
        def decorated_function(*args, **kwargs):
            if 'username' not in session:
                return redirect(url_for('login'))
            if session.get('role') != required_role:
                abort(403)  # Forbidden
            return f(*args, **kwargs)
        return decorated_function
    return wrapper

@app.route('/')
def home():
    return redirect(url_for('login'))

@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        uname = request.form['username']
        pwd = request.form['password']
        user = users.get(uname)

        if user and user['password'] == pwd:
            session['username'] = uname
            session['role'] = user['role']
            resp = make_response(redirect(url_for('dashboard')))
            resp.set_cookie('user_type', user['role'])  # Not secure, for testing
            return resp
        else:
            return 'Invalid credentials!'

    return render_template_string(login_template)

@app.route('/dashboard')
def dashboard():
    if 'username' not in session:
        return redirect(url_for('login'))
    return                          render_template_string(dashboard_template,
user=session['username'], role=session['role'])
```

```
@app.route('/admin')
@role_required('admin')
def admin_panel():
    return render_template_string(admin_template)

@app.route('/logout')
def logout():
    session.clear()
    return redirect(url_for('login'))

@app.errorhandler(403)
def forbidden(e):
    return '<h3>403 Forbidden: You are not authorized to access this page.</h3>', 403

if __name__ == '__main__':
    app.run(debug=True)
```

Output:

```
PS D:\Yenepoya\Semester\Application and Web Security\Practical> python .\b7_cookie_session_secure.py
 * Serving Flask app 'b7_cookie_session_secure' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Restarting with watchdog (windowsapi)
 * Debugger is active!
 * Debugger PIN: 108-349-042
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```