What is Software Security?
Protection of software from unauthorized access, use, modification, or destruction. Goal: Ensure confidentiality, integrity, and availability (CIA triad).

Why is Software Security important?
Growing dependency on software in critical systems.
Increase in cyberattacks exploiting vulnerabilities.
Real-world impacts: data breaches, financial losses, damaged reputation.

Common Threats in Software Security
1. Malware: Viruses, ransomware, spyware.
2. Phishing/Social Engineering.
3. Vulnerability Exploits: SQL injection, buffer overflows.
4. Supply Chain Attacks: Hackers target a trusted third party (like a software vendor or update server) to compromise many users at once.

Practices in Application Security
Annual penetration tests and vulnerability assessments.
Focus often on compliance rather than prevention.
Vulnerability Assessment: Identify and quantify vulnerabilities.
Penetration Testing: Simulate real-world attacks.

Practices in Application Security
Annual penetration tests and vulnerability assessments.
Focus often on compliance rather than prevention.
Vulnerability Assessment: Identify and quantify vulnerabilities.
Penetration Testing: Simulate real-world attacks.

How to implementing S-SDLC
Establish security policies.
Get support from all stakeholders.
Implement incident response plans.
Train users and developers.
Clients can demand security via SLAs.

S-SDLC Steps
Requirements
Designs
Coding
Testing
Deployment

Threat Modeling Basics
Definition: Identify and prioritize potential threats.
Frameworks: STRIDE, DREAD
Example: Threats in a banking application.

Secure Coding Best Practices
Don't hardcode credentials.
Validate all input data.
Use parameterized queries (prevent SQL injection).
Keep dependencies updated.
Vulnerability Management
Use scanning tools: Nessus, OWASP ZAP.
Maintain patch management policies.
Conduct regular security audits.

Key Principles of Software Security
Confidentiality: Info is safe from accidental or intentional disclosure
Integrity: Info is safe from accidental or intentional modification
Availability: Info is available for authorized users when needed.

What are the Key Challenges in Software Security
Rapid DevOps/Agile cycles.
Complex software ecosystems.
Limited awareness and training.
Insider threats.

Common OWASP 10 Vulnerabilities
1. Injection (SQL, command, etc.) 4. Security Misconfiguration
2. Broken Authentication 5. XSS
3. Sensitive Data Exposure

Software Security vs. Application Security

Aspect	Software Security	Application Security
Approach	Preventive	Reactive
Focus	Build security in	Patch after deployment
Scope	OS, Middleware, Applications	Application only
Strategy	Holistic & long-term	Issue-based & short-term

Impact of Security Breaches
Financial loss
Reputation damage
Legal issues & lawsuits
Regulatory fines (e.g., GDPR)
National security risks

Cost of fixing bugs
Cost increases drastically when bugs are found later in the SDLC.
Fixing early is cheaper and reduces operational vulnerabilities.

Security Testing Types
SAST
DAST
IAST
Mix automated and manual testing.

Regulatory and Compliance Requirements
GDPR: Data protection (EU).
HIPAA: Healthcare data.

Aspects of Software Security
Code Level Security, User Input Verification
OS and language differences.
Cryptography.
Access control.
Data-at-rest and data-in-transit protection (e.g., TLS/SSL).

Emerging Trends in Software Security
Zero Trust Architecture (ZTA).
Security in AI-driven applications.
Post-quantum cryptography.

Real-World Security Breaches
SolarWinds supply chain attack.
Equifax data breach.
WannaCry ransomware attack.

Runtime Application Security
Runtime Application Self-Protection (RASP): Real-time protection against attacks.
Memory protection (ASLR - Address Space Layout Randomization, DEP - Data Execution Prevention).
Sandboxing and container security.

Cryptography: Protecting data through encoding.
Goal: keep information from those who aren't supposed to see it
Cryptanalysis: Breaking ciphertext without knowing the key.
Cryptology: The study of both cryptography and cryptanalysis.

Security Requirements in Cryptography
Confidentiality: Data should only be visible to authorized users.
Integrity: Data must not be altered during transmission.
Authentication: Verifying the identity of users or systems.
Authorization: Determining access rights to resources.
Auditing: Tracking actions for accountability.
Non-repudiation: Ensures that a sender cannot deny sending a message.

Types of Security?
1) Unconditional security: Cannot be broken, regardless of computational power. (e.g., One-Time Pad). 2) Computational Security: Can theoretically be broken, but would take an impractical amount of time and computing power. (e.g., AES, RSA)

Cryptographic Terminology
Plaintext: Original readable message; Ciphertext: Encrypted, unreadable message. Key: Information needed for encryption/decryption; Cipher/Function: The encryption algorithm itself; Key Distribution: Method of securely sharing keys between parties.

Symmetric Key Encryption
Pros: Fast and efficient. Cons: Key management is difficult (scales poorly with more users), Key distribution must be secure, Provides confidentiality only (no authentication or non-repudiation).

How Cryptography Works
Uses a well-known encryption algorithm that takes two inputs: 1) plaintext 2) Key. The algorithm scrambles data into ciphertext. Only users with the correct key can decrypt it back to plaintext.

Characteristics of a good cipher
Confusion – Interceptor should not be able to predict how changing one character in plaintext will change the ciphertext.
Diffusion – The characteristic of distributing the plaintext over the entire ciphertext.

Types of ciphers 1) Stream Cipher
Encrypts data one bit/byte at a time. Pros: Fast, low error propagation. Cons: Less diffusion, more vulnerable to modifications.
2) Block Cipher: Encrypts fixed-size blocks of data (e.g., 64 or 128 bits). Pros: High diffusion, immune to insertion attacks. Cons: Slower, higher error propagation.

Brute Force Attack
An attacker systematically tries all possible keys until the correct one is found. Mitigation: Use long, complex keys to make brute-force infeasible.

Asymmetric (Public Key) Encryption Pros: Solves key distribution problem; Cons: Slower compared to symmetric encryption.

Pretty Good Privacy (PGP)
A hybrid system combining both symmetric and asymmetric cryptography: Uses asymmetric encryption to exchange symmetric session keys, Uses symmetric encryption for the actual data transfer (faster), Used in emails and file encryption.

Hash Functions
A hash function generates a fixed-size digest from variable-length input data. Properties: One-way (cannot retrieve original input), Produces unique output for each input, Deterministic (same input → same hash). Ex: MD5, SHA-1, SHA-256, SHA-512.
Salted Hashing: Adds a random value ("salt") to passwords before hashing to prevent dictionary and rainbow table attacks.
Hash collisions: Occurs when two different inputs produce the same hash output.
Rainbow Tables: Precomputed tables used to reverse cryptographic hash functions. Used in password cracking — mitigated by salting.
Applications of Hashing: Password protection, Data integrity (digital signatures), Blockchain verification (linking blocks securely).
Quantum cryptography: can break classical algorithms like RSA. To prevent it uses quantum mechanics (Ex: quantum key distribution) to build systems resistant to such attacks.

TLS
TLS (Transport Layer Security) is a cryptographic protocol that provides secure communication over networks such as the Internet. It evolved from SSL (Secure Sockets Layer) and is used to ensure: Confidentiality, Integrity, Authentication.
Without TLS: Others can see your data (loss of privacy). Data can be altered in transit (loss of integrity). No authentication
TLS

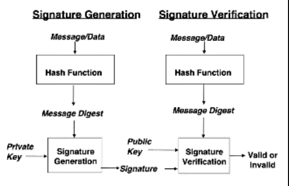
TLS and Load Balancers
Load balancers can handle TLS termination (decrypt/encrypt). Ensures session stickiness or session replication across servers.
Modes: 1) SSL Bridging/Terminating: Load balancer decrypts traffic. 2) SSL Tunneling/Pass-through: Traffic remains encrypted end-to-end.

Public Key Infrastructure (PKI): PKI helps verify the identity of entities communicating securely over a network. Uses public/private keys for encryption. Relies on trusted third parties, called Certificate Authorities (CAs). Guarantees the authenticity of public keys through digital certificates.
Large-Scale PKI Implementation:
(1) HSM (Hardware Security Module): Dedicated device for managing digital keys securely. Provides tamper-proof and tamper-evident features. Expensive but offers strong protection. (2) Directory Server: Stores certificate-related data. Used by CAs and OCSP responders. Optimized for fast lookups (unlike databases). Guarantees the authenticity of public keys through digital certificates.

Protection at Two Levels
1. Lower Level (Channel Protection): Focuses on securing the communication channel, Ensures confidentiality and integrity of data during transmission, Uses encryption and secure protocols, Ex: TLS/SSL, VPNs, IPSec
2. Application/User Level: Works at the software and user interaction layer. Focuses on authentication, access control, and data protection. Prevents application-level attacks. Ex: MFA, RBAC, WAF, Secure database storage and encryption
Attacks on SSL/TLS: Man-in-the-Middle (MITM), Protocol Downgrade (forcing TLS 1.0/1.1), Heartbleed (OpenSSL memory leak), BEAST / POODLE (CBC-related vulnerabilities), Certificate Forgery / Compromised CA
Mitigating TLS Risks: Keep systems and libraries updated, Use the latest TLS version (1.3), Use strong cipher suites with large key sizes, Carefully assess legacy system compatibility, Document risks and mitigation steps.

TLS Handshake Protocol
1. Client Sends Hello, Cipher Suite, & Client Random
2. Server respond back by sending the server random & SSL certificate (Public Key)
3. Server Authentication: The client verifies the server's certificate using the CA's public key (to confirm identity).
4. Client generates a session key and encrypt with server's public key. This is now sent to the server.
5. The server now gets this encrypted message and decrypt with it's private key.
6. Session Key Creation: Both sides now use the shared secret key and use this symmetric key for data transfer.
TLS Security Mechanisms: (1)Privacy: Encrypts messages using symmetric encryption (after handshake). (2) Key Exchange: Uses public key cryptography to exchange a session key. Common algorithms: RSA, Diffie-Hellman. (3) Integrity: Adds a Message Authentication Code (MAC) (e.g., SHA-1, MD5). (4) Authentication: Verifies identity via digital certificates (optional for client).

How digital signature generation and verification works?



Digital Certificates
A digital certificate binds an entity's identity to its public key.
Certificate Creation Process:
(1) Generate a private key. (2) Create a Certificate Signing Request (CSR). (3) Send CSR to a CA. (4) CA validates and issues a signed certificate. Ex of CAs: GoDaddy, Let's Encrypt, Verizon.
Self-Signed Certificates:
Signed by the entity itself, not by a CA. Common in testing or internal systems, but not trusted by browsers. Ex command-line tools: keytool, openssl.

How PGP works?
(1) Generate session key. (2) Encrypt data using session key. (3) Encrypt session key with receiver public key. (4) The receiver gets both of these two. (5) Receiver decrypts session key with private key to get session key. (6) This session key is used to decrypt data by the receiver.

Cookies Attribs: Name: Identifier for the cookie; Content: Stored data (session ID, user token); Domain: The website domain the cookie belongs to; Path: Specifies where the cookie is valid; Expiry: Defines when the cookie expires (optional).
Types of Cookies: Persistent Cookie: Stored even after closing the browser (e.g., "Remember Me"); Non-persistent (Session) Cookie: Deleted when the session ends; HttpOnly: Not accessible via JavaScript → prevents XSS attacks; Secure: Sent only over HTTPS connections.
Cookie based attacks: Session Hijacking; Stealing session cookies;
Cross site request forgery (CSRF): A type of attack where a malicious website tricks a logged-in user into performing unwanted actions on another site (like transferring money or changing passwords). It exploits existing sessions using the user's valid cookies.
Preventing CSRF Attacks:
1. CSRF Tokens / Synchronizer Token Pattern: Tokens are random values generated by the server and embedded in forms or requests to verify the source.
2. Double submit cookies: The token is sent in both a cookie and a form field/header. The server checks if both values match. Prevents CSRF without requiring server-side storage.

XSS Prevention: (1) Input validation; (2) Output sanitization
XSS Types: (1) Reflective XSS: The malicious script is reflected off the server immediately (e.g., through a URL parameter); (2) Stored/Persistent XSS: The script is permanently stored on the server (e.g., in a database or comment field) and executes whenever a user visits the page.
What XSS can do: Session hijacking; Site defacement; Site redirection/Phishing; Data theft; Keystroke Logging
SQL Injections: Injection flaws in SQL can be due to flaws in SQL, LDAP, or OS injection vulnerabilities. SQL Injection can happen in: GET Requets, POST Requets
Impact of SQL Injection: Login bypass; Data theft; Database destruction; Privilege escalation
SQL Injection Types: Boolean-Based (Uses true/false conditions to infer data); Union-Based (Combines results from multiple queries); Error-Based (Exploits database error messages to extract information); Time-Based (Use the SQL SLEEP function to test injection success silently.)
Preventing SQL Injection: Sanitization of User Input. Always sanitize on the server side; Use parameterized Queries (Prepared Statements); Principle of Least Privilege
Sanitization Methods: Blacklisting (Remove known dangerous characters); Escaping (Add escape characters before special symbols); Whitelisting (Accept only known safe inputs)

Security Misconfiguration: Happens when servers, databases, or frameworks are not securely configured. Ex: Default admin credentials; Exposed error messages; Unused services or open ports; Misconfigured SSL certificates; Mitigation: Change default credentials and ports; Disable unnecessary features; Regularly patch and harden configurations.
Insecure direct object reference: Occurs when internal objects (user IDs, file paths) are exposed without authorization.
Insufficient Logging and Monitoring: The system fails to detect or alert on suspicious activities. Impact: Attackers can act for long periods unnoticed. Mitigation: Centralize and monitor logs; Implement intrusion detection systems (IDS); Track failed logins and privilege changes;
Unvalidated Redirects and Forwards: User-controlled input is used in redirects. Ex: Redirects users to a phishing site. Mitigation: Validate and whitelist redirect URLs; Avoid using untrusted user input in redirects.
Missing Function-Level Access Control: APIs or functions are accessible without proper role checks. Ex: /admin/deleteUser accessible by a normal user. Mitigation: Verify authorization for every server-side request; Implement role-based access control (RBAC).
Insecure Deserialization: Occurs when untrusted or modified data is deserialized, possibly leading to remote code execution (RCE). Ex: Attacker sends a malicious serialized object that executes code. Mitigation: Avoid deserializing untrusted data; Use integrity checks or signatures; Employ safe data formats (e.g., JSON).
XML External Entities (XXE): Vulnerability in XML parsers that allows reading of local files or SSRF attacks. Mitigation: Disable external entities in XML parsers; Prefer JSON over XML; Follow secure coding standards

Using Components with Known Vulnerabilities: Applications depend on third-party libraries with security flaws. Ex: Outdated frameworks or OpenSSL (Heartbleed). Mitigation: Regularly update dependencies; Use dependency scanning tools (OWASP Dependency Check, Snyk); Maintain a Software Bill of Materials (SBOM).

Responsible Vulnerability Disclosure: The ethical process of reporting security issues to organizations before public disclosure. (1) Identify and document the issue. (2) Report privately to the vendor. (3) Wait for a buffer period for patch release. (4) Publish as a security advisory.

Secure Software Development Lifecycle (S-SDLC): Security must be integrated from the start of the development process. Key Practices: Static Code Analysis – detect vulnerabilities before release. Dynamic Application Testing – test for real-world attack behavior. Dependency Scanning – monitor for vulnerable libraries. Automation (DevSecOps) – use CI/CD tools like Jenkins, GitHub Actions. Security Reviews – have dedicated teams review code and architecture. Developer Training – ensure awareness of OWASP and secure coding. Security Guidelines – follow frameworks like WSO2 Secure Engineering.

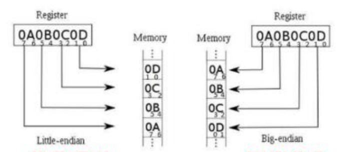
What is Authorization/Access Control: Mechanism using with a system grants permission to access/modify some data or perform some action.
Categorization by Implementation: (1) Logical Controls. (2) Administrative Controls. (3) Physical Controls.
Categorization by time continuum: (1) Directive Controls: Designed to establish desired outcomes. (2) Deterrent controls: Designed to discourage an attacker. (3) Preventive controls: Designed to prevent an attack from happening. (4) Compensating controls: Designed to compensate for a risk that's too difficult to address. (5) Detective controls: Designed to detect an attack while it is happening. (6) Corrective controls: Designed to mitigate the damage once an attack/intrusion has taken place. (7) Recovery controls: Designed to bring back a system to its normal operational state.
Categorization by Enforcement: (1) Discretionary Access Controls (DAC): Owner of the data determine privileges. (2) Mandatory Access Controls (MAC): Defined by System or as per Organizational Security Policy. (3) Non-Discretionary Access Controls: Defined by the Administrator.
OAuth vs OpenID Connect
Oauth uses two step process to authentication (get the token, use the token to get the user data). Open ID connect does this in a single step

Data Access Control Methods: (1)ACL, (2)Access Control Matrix: Same thing as ACL in a form of table, (3)Rule based Access Control: Access Control based on rules, (4)Role based Access Control: Access Control based on user role. (5) Content Dependent Access Control: Based on actual content rather than role. Can be used in environments when roles change frequently. (6) Constrained User Interface: Limit functions user have access to. (7) Capability Tables: Match Subjects with their capabilities. (8) Temporal (Time-Based) Isolation: Constrain access on Time.
OAuth (Grant types)
Auth Code Grant type: (1) Client –> Server side web app (redirection url); (2) Client - > Auth server → backchannel call; (3) Auth server remembers the consents; (4) Request params (refer the spec); (5) Register client app (developer portal); (6) Facebook apps;
IAM Patterns
Password credentials grant type: (1)User's credentials are passed; (2) Client has to be trusted.
Client credentials grant type: (1) Not relevant for sharing user details; (2) About sharing a shared resource in a trusted manner (with authorization)
Refresh Tokens: Supported only in auth code and password grant types

OAuth Usecase
e.g. upload files to Google drive – Auth code
e.g. Google map api – client credentials

Token introspection: Defines the token validation process by the resource server and auth server; Token introspection endpoint in AuthServer; Timestamp skew;

Insecure Deserialization: is a vulnerability that occurs when untrusted data is deserialized (converted back into objects) without proper validation, allowing attackers to craft payloads that cause unexpected behavior — from logic abuse and data tampering to remote code execution (RCE).
Clickjacking: is an attack where an attacker tricks a user into clicking something on a legitimate site by hiding that site inside a transparent or disguised layer (for example an <iframe>), so the user actually interacts with the victim site without realizing it. Prevent: Send the header X-Frame-Options: DENY or SAMEORIGIN.
Buffer overflow: A buffer overflow occurs when a program writes more data to a buffer than it can hold, overwriting adjacent memory locations.
How it works? When input exceeds allocated memory space: Adjacent variables or control data (like return addresses) get overwritten. May lead to program crash (segmentation fault) or arbitrary code execution.
Detection: Operating systems may detect illegal memory writes and raise a segmentation fault to stop the program.
Damage: Causes denial of service (DoS). Can be escalated to remote code execution (RCE).
Big-endian: Most significant byte stored first.
Little-endian: Least significant byte stored first.



Stack Frame: Each function call creates a stack frame containing: (1) Local variables; (2) Return address; (3) Function parameters.
Stack Buffer Overflow (Stack Smashing): Occurs when a buffer on the stack is overwritten. Exploitation Methods: (1) Overwriting local variables. (2) Overwriting the return address so execution jumps to attacker code. (3) Overwriting function pointers or exception handlers.
Heap Overflow: Different from stack overflow — targets heap memory, which is dynamically allocated at runtime. Exploitation: Corrupts internal heap structures (like linked list pointers or metadata); Overwrites function pointers or memory allocation metadata to redirect execution.
Canonical Technique: Overwriting malloc metadata to manipulate subsequent allocations.
Read Overflow (Information Leakage): Occurs when reading beyond buffer limits, exposing sensitive data. Ex: Heartbleed Bug (2014)
Countermeasures: Libsafe Project: Re-implements unsafe C functions like strcpy, strcat, gets with safe versions; StackGuard: Compiler-level defense that inserts "canary" values before return pointers; detects modification before returning; Bounds Checking: Ensures data fits within allocated buffer; DEP & ASLR: Modern OS-level protections: prevent code execution in data memory regions and randomize memory layout.
Remote Code Execution (RCE): Occurs when an attacker successfully runs arbitrary code on a target system, often using buffer overflows or injection flaws.
Difference between stack overflow and heap overflow: Stack overflow overwrites a function's current stack frame (often the saved return address) and hijacks control on return, while heap overflow overwrites adjacent heap objects or allocator metadata and hijacks control later when those corrupted structures are used.

Security in SDLC:
Requirements – Security Requirements;
Design – Threat Modeling, Security Architecture Design Reviews;
Coding – Static & Dynamic Analysis, Code Reviews;
Integration, Validation, Production: Security Testing, Penetration Testing, Secure Configurations

SDLC Approaches: Microsoft SDL, OWASP CLASP, Garry McGraw's Touchpoints.

IAST: An IAST Agent is a software component that hooks into the application at runtime to observe: 1) user input; 2) API calls and internal function execution; 3) Data flow through variables, queries, and file paths; 4) Control flow logic and execution context.

Where IAST Used: QA/Staging; CI/CD; UAT; Production

Secure architecture principles:
Least privilege: Every component should operate with the minimum privileges necessary to perform it's tasks;
Defense in Depth: Multiple layers of security controls should be implemented to protect the system. If one layer fails others can provide additional protection. Reduce likelihood of a single point of failure.
Separation of duties: Critical tasks should be divided among different individuals or systems to prevent conflict of interest and reduce the risk of fraud or error.
Fail secure defaults: Systems should fail in a secure state. If a system fails it should not grant access or disclose sensitive information.
Benefit: Prevents attackers from exploiting system failures.
Example: If an authentication system fails it should deny access rather than allow unrestricted access.
Secure Defaults: Systems should be configured securely by default, with the most secure options enabled out of the box.
Complete Meditation: Every access to resources should be checked for proper authorization; Prevent privilege escalation.
Security by design: Both hardware and software should be configured securely.
Open design: Security should not rely on the secrecy of design or implementation; The system should remain secure even if the design is known.

Terminology:
Assets - a resource of value. May be tangible or intangible.
Threat Undesired act that potentially occurs causing compromise or damage of an asset.
Threat Agent - Something/someone that makes the threat materialize.
Vulnerability - Weakness that makes an attack possible
Attack - Act of malicious threat agent. Also known as Exploit.
Safeguard (Countermeasure) - address vulnerabilities (not threats directly) Ex - Application Design, Writing Secure Code, deploy with least privilege

Threat Modeling: Best performed during the application design phase. Because it's easy to make application changes. Less costly than adding migrations and testing them after code has been implemented and onwards. Advantages: Can be used to find threats early in dev process; Can be used by both security experts and non security experts; Can be used to guide other security assessment activities; Disadvantages: Upfront costs (training, software and setup).

Threat Analysis – Attack Tress: Is a tool to evaluate the system security based on various threats. The root tree represents a security event that can potentially damage an asset. Each attack tree enumerates the ways that an attacker can cause an event to occur.

Security profile:

Category	Consideration
Input validation	Can data in the database be trusted?
Authentication	Are strong passwords enforced?

STRIDE: Spoofing; Tampering; Repudiation; Information Disclosure; Denial of service; Elevation of privilege.

Rank the threats – DREAD Model
Damage potential: How great is the damage if vuln exploited?
Reproducibility: How easy is it to reproduce the attack?
Exploitability: How easy it is to launch an attack?
Affected users: As a rough percentage how many users are affected?
Discoverability: How easy is it to find the vuln?

Threat modeling steps: 1) Define security requirements; 2) Creating an application diagram; 3) Identifying threats. 4) Mitigating the threats; 5) Validating that threats have been migrated.

Blockchain
1. **Blockchain**: Blockchain is a distributed, decentralized digital ledger that records transactions securely and transparently across multiple computers in a network. Once data is recorded in a block, it's nearly impossible to change or delete, making it trustworthy and tamper-resistant. Each block is secured using cryptographic algorithms (like SHA-256 hashing) that link it to the previous block.
2. **Blockchain vs bitcoin: blockchain**: technology, bitcoin: A cryptocurrency (digital money) that uses blockchain to record transactions.
3. **Trusted Arbiter**: A Trusted Arbiter (TA) is a third-party authority that all participants in a system must trust completely to manage, verify, and record transactions correctly (like a bank).
4. **Cons of TA**: Central point of failure, Concentration of Power (complete control over transactions & records).
5. **Solution is blockchain**: distributed system (each node keeps a copy of the entire ledger), don't trust anyone (trust is build through consensus mechanisms), every node has the same transaction history (agree on history).
6. **Blockchain types:**
Public: open to all, reads/writes by all participants, consensus by proof of work (Miners compete to solve a complex mathematical puzzle using computing power. The first one to solve it "proves" they did the work and earns the right to add the next block.)
Private: participants from one organization, write permissions centralized, reads may be public or restricted, multiple algorithms for consensus.
Consortium: multiple organizations, writes requires consensus of several participants, reads may be public or restricted, multiple algo for consensus.
7. **Blockchain:** A technology where information is stored in blocks. Each block is connected to the previous one via cryptography (public key infrastructure, private & public keys). A ledger is the entire collection of all blocks linked together. And this ledger is shared and replicated across multiple computers communicating in P2P manner (decentralized network). Each computer is known as a node. Whenever a new block is added all nodes agree that transactions are valid vis consensus mechanism. Once data is added, it cannot be altered or deleted (prev. tampering)
8. **Block of fact(s):** can transactions, health records, private identity, work history etc.
9. **Chain of blocks:** each block is connected to the previous one. When new transactions are created they are first known as unconfirmed transactions (will be placed in a memory pool). Miners residing in nodes then collect some of the transactions and try to validate the trans. Then they create a block with these data and compete with each other to see who publishes the block first. To decide tis, a consensus algorithm (like proof of work) is used. And this process is called mining. Confirmed block is sent to all other nodes. They check whether the block is correct by consensus and add it to their chain.
10. **Double spending**: happens when trying to spend the same bitcoin twice for two dif payments here, only one is valid.
11. **Rouge node:** A rogue node is a compromised or malicious participant in the blockchain network that tries to disrupt, deceive, or manipulate the system
12. 51% attack: A 51% attack happens when a single entity (or group of attackers) gains control of more than 50% of the network's total computing power or mining power. Happens specialy with (PoW & PoS consensus).
13. **Drawbacks:** complex, slow, wastage of resources, privacy (publicly visible, permanent, accessible to all).
14. **What is 3D secure**: 3D Secure is an online payment security protocol that verifies the cardholder's identity through a password or OTP before approving the transaction. If a card does not support 3D secure, it's declined. 3 parties/domains take part in

14. Emerging trends in software security: 1. **VM vs Docker:** Both VMs and Docker Containers allow multiple applications to run on the same physical hardware. But, (VM runs on a hypervisor/docker runs on docker engine), (VM includes guest OS for each app/docker shares the host OS), (VMS are heavy/Containers are lightweight), (Strong isolation/less isolated than VM). **2. Cloud native apps:** designed to take full advantage of cloud environments (scalable, resilient, easy to update). **3. Security risks in containerized env:** Shared host kernel increases attack surface, insecure base images and outdated dependencies, image sprawl (when there are too many vms hard to track vulnerabilities), Runtime risks- Container escape: An attacker who has code execution inside a container breaks out of the container's isolation and gains access to the host system or other containers.-privilege escalation: An attacker with some access gains higher privileges than intended. **4. Docker security best practices:** use minimal base images (fewer dependencies, smaller attack surface). Eg: Alpine, scan images regularly Eg:Trivy, Clair, avoid running as root in containers (full system privileges. If an attacker breaks out of that container, they could have the full control), use read only file systems (containers cannot modify at runtime), isolate networks and restrict ports (only expose 80/443 for web apps). **5. Kubernetes:** Kubernetes is an open-source container orchestration platform that automatically manages the deployment, scaling, and operation of containers (like Docker containers) across a cluster of machines. **6. Kubernetes architecture**: Two main parts: Control Plane/Kubernetes master (manages the cluster), Nodes/worker nodes (run application workload Control Plane: Api server: entry point for all commands, etcd: distributed key-value database that stores cluster configuration, states, and metadata. It's like Kubernetes' memory, scheduler: assigns pods to available nodes (A Pod is the smallest and simplest deployable unit in Kubernetes. It represents one or more containers that are logically grouped together and share the same network and storage.), controller manager: Ensures the cluster state matches the desired state (what the user wants). Nodes: Kubelet: agent that communicates with the master (api server), kube-proxy: handles network routing inside the cluster (grp of nodes), container runtime: engine that runs containers, Plugin Network (e.g., Flannel, WeaveNet, Calico): ensures that all Pods across different Nodes can communicate securely and consistently, even though they're on different machines, cAdvisor: monitoring tool built by google. **7. Security challenges:** misconfigurations and excessive privileges, poor secrets management (unencrypted etcs), namespace and resource isolation issue, complexity. **8. Security controls:** enforce RBAC (role based ac) & least privilege access, apply pod security standards, define network policies (deny-all), audit logging (Falco, Kube-Bench) 8. Serverless computing: Serverless computing is a cloud model where you can run your code without managing servers (cloud provider automatically provisions servers, scales up/down, charge only when they're running). **9. Serverless security threats:** Function event injection (functions are run by events), over-permissioned IAM roles, cold start latency (when a function is triggered after being idle, the cloud provider must setup everything again — this causes delay) & monitoring blind spots (might miss early-stage attacks), third-party dependency risk (rely heavily on external libraries and npm/pip package), limited observability. **10. Securing serverless architectures**: enforce least privilege IAM (only needed privileges), enable detailed logging and monitoring, use WAFs (web app firewall: a security filter that sits between your users and your web app or API, prevents SQL injection, DDoS)/API gateway throttling (limiting how many requests users or clients can send to your API per second or per minute: prevent DDoS attacks, unexpected costs), validate inputs & scan dependencies, use secure secret stores. 23 **Attacks on ML:** evasion (craft malicious input that look benign), poisoning (injects corrupted data to the dataset), model stealing (repeatedly query a deployed model (via an API) to reconstruct or replicate its internal logic), risks (bypass detect models). **24. Defensive techniques against attacks**: Model Training with Noisy Inputs, Gradient Masking (hide) and Preprocessing Filters, Ensemble Models for Resilience. **25. Securing ML models & pipelines:** Ensure Data Provenance (check data origin and whether altered) and Feature Validation, encrypt model files & secure access, protect inference apis (models exposed via APIs) with rate limiting & logging. **26. Federated learning**: ML is trained locally on devices (at edge) and only updates are sent to the central server. However, data poisoning may occur (GBoard) **27. Differential privacy:** adds random noise to data or results (Apple ios). 28. Zero-trust architecture: based on the idea of "never trust, always verify". Least privilege, continuous verification, device verification. Eg: Google BeyondCorp. 29. Secure Access Service Edge (SASE): combines networking (like WAN or VPN) and security-as-a-service in one cloud-delivered platform. Components: SWG (Secure Web Gateway): Protects users from malicious websites, ZTNA (Zero Trust Network Access), CASB (Cloud Access Security Broker): Monitors and controls use of cloud apps, FWaaS (Firewall as a Service): Cloud-based firewall for network traffic filtering. Applications: Secure remote work access, Branch office connectivity, Controlling cloud app usage

Authorization continuation
Grant types: authorization code, implicit, resource owner password credentials(client collects credentials and exchange them for tokens), client credentials.
IAM Patterns & Provisioning (authorization code gt): JIT Provisioning: Automatically create an app account the first time a user logs in via SSO/IdP, Account association: Link multiple external identities to a single application user.

Open ID Connect
identity layer built on top of OAuth 2.0 that allows users to log in securely using a trusted identity provider (like Google, Microsoft, or Facebook). We don't use client credentials here.
OAuth 2.0 only handles authorization (what a user/app can do).
OpenID Connect adds authentication (who the user is)
Enables Single Sign-On (SSO) across multiple systems.

Authorization code grant type (Client is re directed to auth server (google) and authenticated, Server ends a short lived authorization code which the client exchanges with the backend for an access(access APIs)/ID(JWT: prove identity) tokens

Implicit grant type: authentication server directly returns the access[traditionally] (or ID) token[get these in openID]. No backend involved. Less secure)

Encryption & decrypt: header & payload is hashed and encrypted with an algorithm by the auth server. When this is received by the resource server it is decrypted with the public key of the auth server. Recipient also generates the hash val (if identical, it is valid)

Scopes in OpenID: user data an application can access. OIDC has predefined scopes.

OAuth 2.0 grant types: Authorization code, implicit, hybrid, client credentials.

Nonce: a random, unique string sent with login requests & returned with the ID token to prevent replay attacks (attacker reusing a stolen ID token). This nonce is validated at client's end.

API security: Throttling (limits requests), versioning (manages multiple API versions), federated authentication(API integration with

IOT
A vast network of physical objects (sensors, appliances, vehicles, machines) that connect to the Internet or local networks to collect, share, and act on data. Key traits: connected comms, onboard sensing/compute/radio, and often autonomous operation. Examples span smart home, wearables, industrial and healthcare. Why it matters: huge efficiency gains but massive privacy/security exposure due to scale.
Why IoT security matters
* Scale: Billions of devices → each is a potential entry point.
* Sensitivity: Collects personal/health/financial data → breach impact multiplies.
* Weak baselines: Defaults, no encryption, poor update paths are common.
* Physical consequences: Compromise can unlock doors, alter cars, impact power grids.
* Lifecycle gaps: Devices live for years with no patches → long-term exposure.
* Case: 2016 Mirai botnet hijacked cameras/routers to launch massive DDoS.

1. **Key characteristics:** Connected to internet, minimal human intervention, use sensors, processors and communication hardware. 2. **Pros:** improved efficiency, automation and decision-making. 3. **Cons:** security and privacy challenges. 4. **Why security is important?** Massive device connectivity: therefore each device is a potential entry point to an attacker. Sensitive data exposure: personal, health and financial data. Weak or no security controls: use default p/w, lack encryption, no updates. Real-world impact: can cause physical harm. Lack of updates and lifecycle management: no patching or security updates. 5. **Security challenges in IoT:** Resource-constrained devices (limited cpu etc.), diverse hardware platforms, no common standards, lack physical or software security, prevalence of default passwords & insecure bootloaders, infrequent or no updates, data privacy. 6. **Threats in IoT**: Botnets: attackers attack large no of IoT devices to form a network of "bots" that can launch massive DDoS attacks. Network-based attacks: Spoofing: a malicious actor fakes their identity by impersonating a trusted source to deceive a target into revealing sensitive information or gain unauthorized access. MITM: an attacker intercepts/alters communication between two users. DDOS. Physical attacks: JTAG access: If an attacker gains physical access, they can use JTAG (debug interface built into chips/circuit boards) to read/write memory, dump firmware, extract cryptographic keys, disable protections (secure boot), or install malicious firmware. Unauthorized firmware modification. Eavesdropping & replay attacks. 7**. Lightweight crypto**: as IoT devices need power, small memory and fast computation. Block Ciphers (e.g., PRESENT, PRINCE, Simon/Speck), Hash Functions (e.g., SPONGENT, PHOTON), Stream Ciphers (e.g., Grain, Trivium).
8. **Secure Firmware Development**: Code Signing to verify authenticity, secure bootloaders, memory protection (buffer overloads), minimal access rights, secure logging for audit trails.
9. **Secure Firmware Updates**: TLS Encryption, Digital Signatures, Block downgrades, recover mechanisms, hash validation for package integrity
11. **Shared responsibility in cloud security**: both you and the cloud prov. Has responsibilities, cloud provider: manage infra, h/w, net. Customer: manage OS, app, user access, data security. **12. Service types**: you/cloud. on premise (app, data, runtime, middleware, os, virtualization, servers, storage, networking), iaas(a, d, r, m, o,v, s, s, n), paas(a, d, r, m, o,v, s, s, n), saas(a, d, r, m, o,v, s, s, n) **13. Postquantum security**: quantum computers can break algo. This is about designing cryptographic algorithms to run on quantum computers & withstand attacks from quantum computers. **14. Quantum threats**: Shor's algo: RSA and ECC (Elliptic Curve Cryptography) (asymmetric) rely on the fact that factoring or solving discrete logarithms is extremely hard for classical computers. Shor's algorithm can factor large integers exponentially faster than any known classical algorithm. Action: replace with quantum crypto. Grover's algo: speeds up brute force attack (symmetric), action: increase key size. **15. Cryptosystems at risk**: RSA, ECC, digital signatures, blockchain depending on this. **16. Post-Quantum Cryptography Standardization**: select and standardize quantum-resistant algorithms. CRYSTALS-Kyber: A lattice-based PQC algorithm for encryption and key exchange, replacing RSA and DH. CRYSTALS-Dilithium: A post-quantum digital signature algorithm, used to verify identities and authenticate messages. **16. Lattice-based cryptography**: Secures data using problems from geometry and linear algebra, not number theory (like RSA or ECC). They deal with problems even quantum computers can't solve. Based on Learning With Errors (LWE: It's a math problem where you're given some equations that have been slightly "messed up" with random errors, and your task is to find the original correct equations) & Shortest Vector Problem (Given a huge grid of points (a lattice), find the shortest non-zero vector (the nearest point to the origin). Efficient (OpenSSL/Chrome experiments). **17. PQA families**: code-based: McEliece (Based on errorcorrecting codes, where encryption hides a message inside intentionally added "errors."), Hash-Based: SPHINCS+ (relies on hash functions, for digital signatures), Multivariate: Rainbow (Based on solving systems of multivariate polynomial equations, a mathematically complex problem, broken recently, digital signs.), Isogeny-Based: SIKE (Uses isogenies between elliptic curves (mathematical mappings) for key exchange.). **18. Hybrid crypto models**: classical encryption (like RSA or ECC) with post-quantum algorithms (like Kyber), smooth transit, layered security. **19. Crypto agility:** The ability of a system to swap cryptographic algorithms easily without major redesign or downtime. **20. Transition:** NIST guidelines, hybrid models, ensure crypto-agility, test PQ libraries. **21. Role of AI/ML**: Automates threat detection & management, fast, fewer false positives, 24/7 monitoring (antivirus, phishing detection, SOC automation, securing ML models). **22. AI for malware detection**: learn patterns in known malware (static & dynamic features) eg: Microsoft Defender ATP, VirusTotal ML Scoring. **23. Anomaly detection**: identifies behaviors that deviate from normal patterns in users, systems, or networks (insider threats, account compromise, techniques: k-Means (Groups normal behavior patterns; any point far from clusters is an anomaly), isolation forest (Randomly isolates data points — anomalies are isolated faster since they're different), autoencoders (Neural networks that learn to reconstruct normal behavior. If reconstruction error is high, it's likely an anomaly))

Mobile application Security
1. **Types of apps:** Web, native, hybrid (wrapped inside a native container). **2. General issues:** Enterprise grade encryption on general devices is not common, hard to ensure physical security, untrusted removable media (chips), jailbreaking and rooting app stores do not validate security of apps effectively, user friendliness over security (tokens with long TTL), small displays = easier to perform phishing attacks. **3. Five pillars:** Traditional access control: passwords & screen locking, Application provenance: tamper resistant (each application is stamped with the author's identity). Therefore, the use can decide whether to use the app based on the author's identity, Encryption: conceal data (if device lost), Isolation: limit access of sensitive data and systems on a device, Permission-based access control: Set of permissions to each app (blocks if exceeds this)
4. **Android(google):** Security model: Android runs on a modified Linux kernel. Every app/process has a separate userid. Each app gets its own process and virtual machine/runtime instance (ART). Crashes or bugs stay inside that sandbox. Permissions/Access Control: Interacts freely with OS for general trans. But should get permission at install time for other ones. Any additional privilege is set in manifest file. Storage: SQLite. Development: Java, compiled into Davlik executable bundled with manifest files. Packaged to Android Package Files (APK) and signed with developer's public key pair and sent to google app store. Open Source (Android Open Source Project) – freely available and customisable. **5. ios (Apple):** Security model: UNIX/BSD roots: iOS inherits security from TrustedBSD; it enforces MAC (Mandatory Access Control) system-wide. Sandboxing (every app runs inside its own container). Use AES-256 to encrypt all data stored in the flash memory. Permission/access control: can access file system allocate resources freely but others need to be permitted by the user. Storage: SQL flat file database + keychains (the OS's secure vault for small secrets.). Development: using Objective-C, bundled with an entitlements and preferences, code signed by an apple issued certificate. **6. Top mobile application issues:** M1; improper platform usage: misuse platform-spec security features or fail to follow platform guidelines (not using ios's keychain). Impact: exploit system features. M2; insecure data storage: storing in plain text, not encrypting SQLite dbs. Impact: if phone is lost, data can be taken. M3;Insecure communication: no encryption during transit (HTTP instead of HTTPS). Impact: Man in the middle ttacks. M4; Insecure authentication: M5;Insufficient crypto: outdated crypto (MD5/SHA-1). Impact: brute force. M6; Insecure authorization: no role based access. Impact: unauthorized access. M7; Client code quality: vulnerabilities, buffer overflows. Impact: crashes, injections. M8; Code tampering: Reverse-engineering APKs or IPA files and redistributing them with malware. Impact: Users install fake or trojanized apps, exposing data and devices to compromise. M9; Reverse Engineering: Attackers analyze the compiled code to extract logic, algorithms, or keys (APKTool, Frida) Impact: reveal secrets. M10; Extraneous Functionality: Test or debug functions in production apps (hardcoded credentials). **8. Mobile app testing:** SAST(MobSF, SonarQube), DAST(simulate real world attacks). Mobile Security Testing Guide (MSTG) (by OWASP: a framework for manual and automated testing on mobile apps). ASVS - Application Security Verification Standard: Security req. & best practices for app dev. Emulator Android Debug Bridge (ADB): simulate real device behavior. Tampering traffic (MITM proxying): Intercepting, viewing, and modifying an app's HTTP(S) requests and responses by routing the device through a proxy (e.g., Burp/ZAP). Verifying the signer: verifying whether the app is signed.