

8. Lecture 8: Low Level Security

1. **Why low level (security):** Securing the layers closest to hardware/OS (machine code, ISA, micro-architecture) gives a trustworthy base; a flaw here can compromise everything above (apps, protocols).

2. **Cross Site Scripting:** An attacker gets a website to run their JavaScript in a victim's browser.

3. **Clickjacking:** My SSD vulnerability. 4. **Unvalidated redirects:** App redirects to a URL taken from untrusted input; attackers use it for phishing or open-redirect chains.

5. **Buffer overflow:** Occurs when a program writes more data into a fixed-size buffer making the extra bytes get spilled into adjacent memory which might overwrite it. This may cause a crash or be abused to change control flow (run attacker's code). **Adjacent memory:** If buf has space for 16 bytes and you write 30 bytes, the extra 14 bytes spill into the next boxes—that's adjacent memory. `strcpy(A, "excessive" sizeof(A))`

Solution: Bounds checking (making sure that you don't read/write past the size of the buffer).

6. **Segmentation Fault:** Sometimes the OS throws this (a fault raised by hardware with memory protection that notifies the OS about a memory access violation) to terminate the process. `gets(password) ->` writes until new line and is removed from modern c as it doesn't know the buffer size. 7. **Where can the data come from (for a buffer overflow)?** Text entered in GUI, Via CLI, Via a program, Via a file, env variables. 8. **Damage:** Can cause DoS due to segmentation error / corruption of other program data, control flow / shutdown of the computer (DoS) / be able to execute code on the computer (Remote Code Execution (RCE)).

9. **Memory Structure: Heap:** Memory allocated during runtime (malloc/new/allocate/free/delete/remove). **Stack:** Store local variables (in an element called frame), **Data (.data):** Read/write region for global/static variables with explicit initial values, **BSS (.bss):** Read/write region for global/static variables without initial values, **Text (.text):** ead-only region containing the program's compiled machine instructions.

10. **Big endian & little endian:** The in which a block of bytes is saved in memory.

Little endian: LSB stored in lowest address / MSB stored in highest address. Big endian: opposite of little one.

REGISTER SET INTEL 80386 ARCHITECTURE		
Category	Register	Function
Data	EAX (accumulator)	Used for arithmetic and data movement. Each register can be addressed as either a 16 or 32 bit value. EBX can hold the address of a procedure or variable.
	ECX (counter)	EDX (data)
	ESI (source index)	EDI (destination index)
	ESI (source index)	EDI (destination index)
Segment	CS (code segment)	Used as base locations for program instructions, data, and the stack. All references to memory involve a segment register used as a base location.
	DS (data segment)	ES (extra segment)
	FS (FS segment)	GS (GS segment)
	FS (FS segment)	GS (GS segment)
Index	ESP (base pointer)	Contains the offsets of data and instructions. The term offset refers to the distance of a variable or instruction from its base segment. The stack pointer contains the offset of the top of the stack.
	EIP (instruction pointer)	Contains the address of the next instruction to be executed within the current code segment.
	EIP (instruction pointer)	Contains the address of the next instruction to be executed within the current code segment.
	EIP (instruction pointer)	Contains the address of the next instruction to be executed within the current code segment.
Control	EFLAGS	Contains the status of the processor. It contains the status of the processor. It contains the status of the processor.
	EFLAGS	Contains the status of the processor. It contains the status of the processor. It contains the status of the processor.
	EFLAGS	Contains the status of the processor. It contains the status of the processor. It contains the status of the processor.
	EFLAGS	Contains the status of the processor. It contains the status of the processor. It contains the status of the processor.

13. **Stack smashing (stack buffer overflow):** Overwrite a func call / return address / function pointer / local variable(or pointer)of a diff. Stack frame. 14. **Difference between stack overflow and heap overflow:** Stack overflow overwrites the 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. 15. **Canonical heap overflow:** you overflow one heap buffer so it corrupts the next chunk's headers (allocator pointers/sizes); then when the program calls free()/malloc(), the allocator uses those fake headers and writes a pointer where you chose, letting you overwrite a sensitive pointer (e.g., a function/table) and take control. 16. **Read overflow:** permit reading past the end of buffer leaking secret info. Eg: Heartbleed bug in OpenSSL (heartbeat messages).

17. **Counter measures:** Libsafe project (re-implemented functions – strcpy(), strcat(), gets()), StackGuard / Stack Canaries: The compiler inserts a random "canary" value on the stack between local variables and control data (saved frame pointer / return address). On function exit, the compiler checks the canary. If an overflow smashed past locals, the canary changes → program aborts before using a corrupt return address.

10. Lecture 10: IoT Security

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, n), paas(a, d, r, m, o, v, s, n), saas(a, d, r, m, o, v, s, n) 13. **Post-quantum 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 error-correcting 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 sigs.), **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)).

9. Lecture 9: Mobile Application Security

1. **Types if 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. **Windows 11 (Microsoft): Isolation:** each app runs in an AppContainer sandbox. **User consent & control:** request explicit consent to access sensitive data (cam, loc, mic etc). **Address space layout randomization (ASLR) & Control flow guard (CFG):** randomizes memory addresses each time the app runs. **Secure boot:** only trusted digitally signed software loads during startup (part of UEFI firmware). Prevents rootkits & bootkits from executing before the OS starts. Works with Defender system to maintain the integrity of the boot process. **Trusted Platform Module (TPM 2.0):** h/w based security chip required for win 11. Provide cryptographic key generation & storage (bitlocker keys, login credentials). Measures & verifies sys integrity during startup to prevent unauthorized changes. 7. **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 attacks. **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.

12. Lecture 12: SSDLC

1. **Bad SE Practices:** Failed projects, lost money, stressed employees, poor customer value. 2. **Good SE Practices:** Successful proj, happy customers, business value, lower stress for developers.

3. **Waterfall model:** short, no changing requirements, precisely documented requirements, stack cannot be changed. 4. **Iterative:** requirements strictly predefined, large-scale projects. 5. **Spiral model:** major edits, unsure requirements, mid-high-level risk projects, risk driven approach. Each phase of the Spiral Model is divided into four Quadrants: Identify Objectives, identify risks, dev and test, plan the next iteration. 6. **V-Shaped:** small, mid-sized projects, requirements strictly redefined. Testing is planned in parallel with each development phase. Each verification stage (like requirements, design) has a corresponding validation stage (like acceptance, system, or unit testing), forming a V shape that ensures early error detection. 7. **Agile:** needs change, flexibility, collaboration, and rapid delivery of small, functional parts of software.

8. **SSDLC approaches: Microsoft SDL:** Training(core security training), requirements(security requirements, quality gates, risk assessment), design(design requirements, analyze attack surface, threat modelling), implementation(approved tools, deprecate unsafe funces, static analysis), verification(dynamic analysis, fuzzing, attack surface review), release(incident response plan, final sec. review, release archive), response(execute incident response plan). **MS SDL:** Efforts are grouped into seven phases: training, requirements, design, implementation, verification, release and response. 9. **Attack surface:** An attack surface is the total number of possible points where an attacker can try to enter, exploit, or interact with a system to cause harm.

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 yms 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 etc), 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).

11. Lecture 11: Threat Modelling

1. **Threat modelling**: process of identifying, analyzing, and prioritizing potential security threats and vulnerabilities in a system before they can be exploited. Best performed in design phase. Less costly than mitig.

2. **Challenges**: time consuming, difficult to show demonstrable ROI, fairly dry stuff to do.

3. **Exploit**: An exploit is a piece of software, code, or technique that takes advantage of a vulnerability (weakness) in a system, application, or network to cause unintended behavior.

4. **Attack tree**: A tool to evaluate the system security based on various threats. The root represents an event that can damage an asset. Each path through an attack tree represents a unique attack. A system can have a forest of attack trees.

5. **Use and misuse cases**: Use cases describe how legitimate users interact with a system to achieve goals. Misuse cases describe how malicious users (attackers) might try to abuse or attack the system <<threatens>> <<mitigate>>. 6. Do not focus on attacker, focus on assets.

6. **Create architecture overview**: define what the application does/how it used & diagram.

7. Decomposing the application: It means breaking the application into smaller components to analyze how data flows, where trust breaks, and what areas are at higher security risk. (Identify trust boundaries (dif trust levels), identify data flow (where data can be tampered), identify entry points(potential attack surfaces), identify privileged code (code with high permissions to add extra security), document the security profile).

8. **Trust boundary**: A trust boundary is the point in a system where data or control passes between two components with different levels of trust. it's where untrusted data enters a trusted system. Eg: user browser & server. 9. **Security profile**: category & consideration (eg: authentication=are strong passwords enforced?)

10. **Identify threats**: threat lists (identify relevant threats from possible threats), STRIDE (identify threats by category), optionally draw threat trees (attack trees).

11. **STRIDE**:

S -> Spoofing (impersonating someone or something), desired property: authentication

T -> Tampering (altering data without authorization), desired property: integrity

R -> Repudiation (ability to claim to have not performed some action against an app), dp: non-repudiation

I -> Information Disclosure (exposure of info to unauthorized users) dp: confidentiality

D -> Denial of Service (Make the service unavailable) dp: availability

E -> Elevation of Privilege (elevate privileges without authorization) dp: authorization

12. **DREAD**: Ranks the threats

D -> Damage potential (how great the damage if vulnerability exploited)

R -> Reproducibility (how easy to reproduce the attack)

E -> Exploitability (how easy to launch the attack)

A -> Affected users (how many users are affected)

D -> Discoverability (how easy it is to find the vulnerability)

12. **Threat modelling steps (MS SDL)**: Define security requirements, create application diagram, identify threats, mitigate threats, validate that threats have been mitigated.

Threat	Mitigation
Spoofing	PIsec. Digital signatures Message authentication codes Hashes
Tampering	ACLS Digital signatures Message Authentication Codes
Repudiation	Strong Authentication Secure logging and auditing
Information Disclosure	Encryption ACLS
Denial of Service	ACLS Quotas High availability designs
Elevation of Privilege	ACLS Group or role membership Input validation

13. **DFD (Data Flow Diagram)**: External entity (rectangle) S&R: people & external systems, Process (circle) STRIDE: code, Data store (two parallel lines) TRID: data at rest (databases/registery keys), data flow (curved up line TID): how data flows, trust boundary (dash line): a point where data flows from one trust level to another.

14. **Identify threats from dfd**: brainstorming, STRIDE method.

15. **Mitigation**: address threats (redesign, use standard mitigations, use unique mitigations, accept risk in accordance with policies) 16. **Validation**: validation of model, enumerated threats, mitigations, assumptions

17. **Advantages**: identify threats early, can be used by both security and non-security experts, can be used to guide other security assessment activities. 18. **Disadvantages**: upfront costs (training, soft. & setup).

19. **Tools**: ThreatModeler, Sea Sponge, OctoTrike, SeaMonster, Microsoft threat modelling tool 2014, securttree

5. Lecture 5: Web Security

1. **Cross Site Scripting**: JavaScript injection, XSS. Makes the victim's browser execute user-supplied code (malicious scripts) inside a trusted website's page. **Prevention**:input validation, output sanitization, use OWASP java encoder, private browsing (avoid cookie exposure), CSRF protection, Implement a CSP header to restrict the sources from which scripts can be loaded. 2. **Types of XSS**: Reflected XSS – script comes from a crafted URL or request and is immediately reflected in response. Stored (Persistent) XSS - malicious script is saved on the server (e.g. comment fields) and delivered to many users later. 3. **What XSS can do**: Session hijacking, site defacement, phishing, data theft, key logging. 4. **SQL injection**: unvalidated user input is inserted into an SQL query and executed by the database. Prevention: server-side sanitization (blacklisting: remove bad chars';-- cuz easy to bypass, escaping: transform special characters so they are treated as data, not SQL syntax, whitelisting: validate), parameterized queries / prepared statements 5. **Types of SQL injection**: Boolean based (true/false reveal info, true-there's a value, false-no value, examine these), union based(Attacker uses UNION SELECT: prev: least privileges), error based(causes the database to produce useful error messages (via crafted inputs), prev: never expose raw DB error messages to users), time based (Attacker injects a conditional that, if true, executes SLEEP(n) (or equivalent) making the web response slower. By checking whether the response is slow or fast, attacker learns whether the condition was true., prev: Rate-limiter).

6. **Eavesdropping / Network Sniffing**: passive attack (no altering) where an attacker intercepts and reads network traffic as it travels across a network, without altering the data. [Use encryption (HTTPS, SSH, TLS, VPNs), avoid public wifi, monitoring tools] 7. **Man-in-the-Middle (MitM)**: an attacker secretly intercepts and can read, alter, or inject traffic exchanged between two parties who think they're communicating directly. The attacker forwards traffic to the recipient. 8. **Session Hijacking**: an attacker steals a valid user session & then uses that session to impersonate the user without needing their credentials.[Short token lifetimes, TLS, MFA]. 9. **DNS Cache Poisoning**: modifies DNS records to direct users to malicious sites[Enable DNSSEC (domain signing + validation: signs records cryptographically), Run up-to-date DNS software & OS patches]. 10. **CSRF**: a malicious website tricks a user's browser into performing unwanted actions on another website where the user is already authenticated [CSRF Tokens (Synchronizer Token Pattern), avoid using GET for state changing requests].

11. **Interception while being transmitted/gain unauthorized access**: Eavesdropping/network sniffing, Man-in-the-Middle (MitM) passive variant, Session Hijacking 12. **Intercepted & altered**: Man-in-the-Middle (MitM) — active variant, DNS Cache Poisoning, Replay attacks 13. **IDOR**: accessing another's data using url manipulation. 14. **Server-Side Request Forgery (SSRF)**: web security vulnerability that allows an attacker to trick a server (not the client) into making requests to unintended locations — internal or external — on behalf of the attacker (use user supplied url).

Other notes

1. **Cipher block chaining**: IV & chaining. Solution for ECB. 2. **TLS Tunneling**: Keeps the entire data stream encrypted end-to-end, even as it passes through the load balancer. 3. **TLS Bridging**: decrypts the traffic at the load balancer (to inspect or route it), which exposes data temporarily 4. **Access control mechanisms**: ACL, Access control matrix, role based access control, content dependent access control, capability tables, temporal(time-based) isolation (constrain access on time), constrained user interface. 5. **OAuth 2.0**: user remain authenticated for extended periods: refresh tokens, if they are stolen, attackers can generate more access tokens. 6. **Inference attacks**: occurs when an attacker combines multiple non-sensitive or seemingly trivial pieces of information to deduce or infer sensitive or confidential data. 7. **Cipher block chaining mode**: parallel decryption of multiple blocks is possible. 8. **PKI**: framework used to manage digital certificates and public-private key pairs to enable secure communication and authentication over insecure networks (certs, CA, RA (verifies identity before cert is issued), asymmetric encr., certificate revocation list), classes: 0: testing, 1:individual email veri., 2: software signing & basic identity confirmation, 3:online tran & identity verification, 4: B2B, 6:high level cert, 7: financial cert, 8: gov cert.

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. 30. **Data protection laws**: GDPR (General Data Protection Regulation – Europe), CCPA (California Consumer Privacy Act – USA) (protect user data). 31. Compliance standards: ISO 27001 (manage & protect info security risks), NIST CSF (guidelines for managing/improving cybersecurity), SOC 2 (standards for service providers).

13. Lecture 13: 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. **Rogue 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 specially 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

10. Lecture 10: Open ID Connect

1. **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.

2. **JWT (JSON Web Token)**: A JS Object that is used to represent user info. 3 components: header (algorithm used), payload (user info: issuer), signature (signed with a secret key or private key). 3. **Authorization code grant type** (Client is re directed to auth server (google) and authenticated, Server sends 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). 4. **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). 5. **Scopes in OpenID**: user data an application can access. OIDC has predefined scopes. 6. **OAuth 2.0 grant types**: Authorization code, implicit, hybrid, client credentials. 7. **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. 8. **API security**: Throttling (limits requests), versioning (manages multiple API versions), federated authentication(API integration with Ips(google)), lifecycle management(controls lifecycle of APIs).

6. Lecture 6: OWASP Top 10

1. **OWASP**: OWASP (Open Web Application Security Project) is a nonprofit foundation that provides free resources, tools, and guidelines to improve software security. 2. **OWASP top 10**: A01 Broken Access Control(access data beyond permission level), A02 Cryptographic Failures(data exposure due to weak encryptions), A03 Injection(untrusted input), A04 Insecure Design(missing security controls), A05 Security Misconfiguration(default p/w, outdated settings), A06 Vulnerable & Outdated Components, A07 Identification & Authentication Failures(missing MFA), A08 Software & Data Integrity Failures(tampering, unsigned code), A09 Security Logging and Monitoring Failures(lack of logging), A10 Server-Side Request Forgery (SSRF)(server makes unintended requests). 3. **Insecure Direct Object Reference (IDOR)**: Occurs when user input directly accesses internal objects (e.g., /user?id=2 shows another user's data). 4. **Insecure Deserialization**: Occurs when untrusted or tampered serialized data is processed by the server. Can lead to remote code execution. 5. **XML External Entities (XXE)**: XXE happens when an application processes XML input that contains a reference to an external entity — and the parser blindly loads it, allowing attackers to read files, expose data, or even perform remote requests.

7. Lecture 7: Authorization (grant/deny permission)

1. **Types of Controls (categorized by implementation)**: **Logical controls (protections by software & systems)**: authentication, firewalls, network segmentation, **Administrative controls(standards, rules, policies)**, Physical controls(tangible measures protecting facilities * h/w). 2. **Control functions/purposes**: **Directive**(established desired outcomes/what to do; policies), **deterrent**(discourage an attacker; warning signs,ctv), **preventive**(prevent an attack;firewalls), **compensating**(compensate when control is impractical; extra monitoring), **detective**(detect attacks in progress; log analysis, IDS), **corrective**(mitigate damage; isolation scripts), **recovery**(return to operation;backups). 3. **Categorized by enforcement**: **Discretionary access controls(DAC)**: owner of data determines, **Mandatory access controls(MAC)**: system/organization, **Non-discretionary access controls**: administrator. 4. **ACL**: For each resource store a list of permissions & which users/grps have them. 5. **Capability list**: For each principal (user) store a list of capabilities (resource + permitted actions). 6. **OAuth**: an authorization framework that lets applications obtain limited access to user resources on another service, typically without sharing credentials. 7. **Grant types**: authorization code, implicit, resource owner password credentials(client collects credentials and exchange them for tokens), client credentials 8. **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. 9. **Token introspection**: An API where a resource server can ask the IdP if a token is valid and get its metadata (active? scope? user? enrvr?). 10. **Dynamic client reistration/client registers itself at IdP via API** setting a client id(and/or client secret), metadata.