Cyber Security

> Course summary - Ahmed Nouralla

Introduction

- **Information Security:** is about information and information systems protection from unauthorized access, use, disclosure, disruption, modification, or destruction in order to provide confidentiality, integrity, and availability.
- **Cybersecurity:** is about protecting systems, networks, and programs from cyber attacks.
- **Cyber attacks:** aim to access, change, or destroy sensitive information; extorting money from users; or interrupting normal business processes.
- **CIA Triad**
  - **Confidentiality:** information should not be disclosed to unauthorized parties.
  - **Integrity:** information should not be modified in an unauthorized manner.
  - **Availability:** system or resource shall be available for its intended use.
- **Legally Defensible Security**: to obtain legal restitution, one must demonstrate that a crime was committed, that the suspect committed that crime, and that reasonable efforts were taken to prevent the crime.
- **Security governance**:
  - Standards/frameworks/guidelines/policies related to defining, supporting, and directing the security efforts of an organization.
  - **Security management:** identifying assets, followed by developing, documenting, and implementing practices to protect those assets, following the governance rules.
  - **Attestation:** the procedure (typically taken by the government) for confirming compliance with information security requirements.
    - They confirm that the objects of informatization (systems or premises) are protected from unauthorized access or leakage due to any reason.
  - **Certification of information security tools**: the procedure of proving the security of a certain tool/system and allowing its use in critical infrastructure systems.
  - **Example:** the security management team implemented procedures/practices and followed the recommended guidelines to satisfy the standards of (ISO 27000 Series, NIST SP 800-53, NIST SP 800-171, COBIT, CIS Controls, GDPR, PIC DSS, PCI PA, ...), and so they got attestation.
- **Data classification**
  - **Military classification:** top secret > secret > confidential -> sensitive -> unclassified.
  - **Commercial classification:** confidential/private > sensitive > public.
- **Security Roles and responsibilities:** Chief Information Security Officer (CISO), senior manager, security professional, data owner/custodian, user, auditor.

Risks and Threats

- **Risk management:** identifying factors that could damage or disclose data, evaluating those factors (based on data value and countermeasure cost), and implementing cost-effective solutions for mitigating or reducing risk.

    - **Risk-based approaches to cyber security** is done by information security department of an organization, it involves risk identification, assessment, and control.
        - How risk management is implemented and how it can be used for decision making totally depends on the organization.
    - **Two main approaches:**

Quantitative Approach	Qualitative Approach
Identifying risk objectively based on statistics and historical data, more complicated, allows to produce cost-benefit analysis in terms of money.	Identifying risk subjectively based on opinions of skilled evaluators, more straightforward, sometimes used along with the quantitative approach.

- **Asset:** anything used in a business process and should be protected.
    - **Asset valuation:** a dollar value assigned to an asset based on several factors (actual hardware costs, costs to develop, maintain, administer, advertise, support, repair/replace, ...)
    - **Vulnerability:** the weakness in an asset or the weakness/absence of its safeguards.
- **Threat:** any potential event that may cause an undesirable or unwanted outcome for an organization or for a specific asset.
    - **Examples:** damage, destruction, alternation, loss, disclosure.
    - **Threat agents** (insiders/outsiders) are driven by some **motivation** to take certain **threat actions (attacks)** and exploit **vulnerabilities**.
    - **Risk:** the possibility that a threat will exploit a vulnerability.
        - Probability of vulnerability exploitation * value of information asset - percentage of risk mitigated by current controls + uncertainty of current knowledge of the vulnerability.
    - **Threat modeling**: the process by which potential threats are identified and enumerated based on security objectives, it also concerns countermeasures prioritization.
        - **Common methodologies:** Microsoft Threat Modelling, OWASP Threat Modelling.
        - **OWASP Threat Modeling process [ref.]**
            - Decompose the Application (i.e., crate Data Flow Diagram).
            - Determine and Rank Threats (e.g., use STRIDE model).
            - Determine Countermeasures and Mitigation (e.g., based on CVSS score): decision can be to accept, eliminate, or mitigate.

Web Application Security

Recall:

- Static vs Dynamic web pages
- Monolith application vs Microservices
- Bare metal vs VMs vs Containers
- Technical stack.
- IaaS, PaaS, SaaS
- Vertical vs Horizontal scaling.
- N-tier/layered architecture

    - 2-tier (split architecture): frontend, backend.
    - 3-tier: database layer, application/business logic layer, UI/client layer.
- HTTP, REST, SSL, HTML, XML, JSON, API, SOA, SOAP, WSDL, RPC, ESB, BFF Gateway, WAF, SSR, CSP, MQP.
- Service mesh, service aggregator.
- Synchronous vs Asynchronous communications.

OWASP Top 10 (2021)

Enumeration of the top 10 most critical security risks to web applications, there exists a similar list for mobile applications.

1. Broken Access Control

- **Description:** attacker was able to access a resource or perform an action that he/she was not supposed to have access to, due to some misconfiguration from admin, or the deployment of a vulnerable system.
- **Examples:** modification of URL or other parameter to gain admin privileges or unauthorized access to some object.
- **Countermeasures:** deny by default policy, access control checks at all layers, code reviews, implementing ownership model, disabling unnecessary functionalities, API rate limit, automatic testing and token validation.

2. Cryptographic Failures

- **Description:** attacker was able to access (stored/cached or transmitted) sensitive data due to the cryptographic algorithm not being implemented correctly or is vulnerable.
- **Examples:** weak hashing/encryption algorithms, weak/compromised/reused keys, or data transmitted over clear text due to the server not enforcing security rules.
- **Countermeasures:** proper implementation and enforcement of encryption depending on asset value on all stages of data lifecycle.

3. Injection

- **Description:** attacker was able to send malicious input to a vulnerable system, the input was processed, allowing the attacker to change the course of execution and manipulate the system into doing an unintended behavior
- **Examples:** leaking/corrupting/destroying sensitive data with SQL/NoSQL injection or other query languages, OS command injection, or CRLF injection to inject HTTP request headers. Payloads can be injected in request line, headers, or body.
- **Countermeasures:** properly sanitize input by escaping special characters and filtering commands or limiting input length, allow using commands only from a whitelist, no direct substitution of user input into code (e.g., process through an isolated API first), handle exceptions and make sure they don't expose sensitive data.

4. Insecure Design

- **Description:** attacker was able to hack into a system because it was vulnerable by design (there was an architectural flaw in the way the system is designed), there was no error from the admin side or the developers who implemented it.
- **Examples:** insufficient elaboration at the stages of analysis and planning didn't define how to handle certain use cases of the application which lead to vulnerabilities.
- **Countermeasures:** implement secure software development lifecycle, use safe design patterns, use threat modeling, data validation at all levels, write tests, limit resource

consumption.

5. Security Misconfiguration

- **Description:** attacker was able to hack into a system because of an error from admin side, as opposed to the previous point.
- **Example:** https not enforced in server configuration, unnecessary features are left enabled, default credentials are not changed.
- **Countermeasures:** automatic configuration management and configuration verification that will reduce the possibility of human errors, removing unnecessary features and frameworks, the use of security headers and directives.

6. Vulnerable and Outdated Components

- **Description:** attacker was able to exploit a vulnerability in a certain component (OS, DBMS, API), or a dependency (software library) used by the target system, due to the component being vulnerable by design or not being updated.
- **Example:** the usage of old version of HTTP webserver.
- **Countermeasures:** regular scanning and updating of application dependencies and hosting systems, removing unused dependencies, obtaining dependencies from official trusted sources, proper monitoring.

7. Identification and Authentication Failures

- **Description:** attacker stole user credentials or was authenticated into a system they were not supposed to access due to the system not implementing authentication correctly.
- **Examples:** the system allows weak passwords, doesn't block automated attacks, doesn't use 2FA, doesn't specify session timeout, or uses an ineffective credential recovery process.
- **Countermeasures:** enforce strong passwords, 2FA, block bruteforce attacks (e.g., CAPTCHA), use strong session manager and specify session timeout.

8. Software and Data Integrity Failures

- **Description:** attacker was able to modify data or software packages due to the system not verifying data integrity properly.
- **Example:** system doesn't validate digital signatures of certificates, or checksums for received software packages; a system that doesn't do these checks can be tricked into de-serializing a malicious object.
- **Countermeasures:** use digital signatures, ensure that packages are grabbed from trusted resources through a secure connection, use a tool for dependency checks.

9. Security Logging and Monitoring Failures

- **Description:** attacker was able to hack into a system undetected because the system lacks extensive and explanatory logging or the logs were not inspected/monitored carefully, or the alerting mechanism was blind to the attack or very slow to report.
- **Example:** Data can be stolen from a system deployed on the cloud, it will take time for the provider to notify the organization.
- **Countermeasures:** implementing the necessary logging, auditing, monitoring, and alerting systems, having a defined incident response process.

10. Server-Side Request Forgery:

- **Description:** attacker was able to abuse the server to access or modify internal resources (or fetch data from untrusted sources), by deceiving the server into communicating with a user-supplied URL. This kind of attacks are critical as they bypass firewalls, VPNs, and ACLs.
- **Example:** the system takes user-supplied URLs and grabs resources from them, an attacker can supply `localhost` or some private IP, this will make the server expose internal data to the attacker, bypassing any defense mechanisms as the response is coming from the server.
- **Countermeasures:**
    - **At the network layer:** internal servers should be separated from the ones fetching remote resource, firewalls should deny by default (only allow essential intranet traffic based on known data lifecycle).
    - **At the application layer:** sanitize and validate all client-supplied input data, fetch resources only from trusted URLs through HTTPS, don't send raw responses to client.

Some OWASP Projects

- **Zed Attack Proxy (ZAP):** tool for checking the security of web apps (scanning, fuzzing, automation).
- **Juice Shop** (vulnerable web app that uses ExpressJS, NodeJS, Sequelize+SQLite, AngularJS, Material UI).
- **bWapp** is a similar vulnerable-by-design app.
- **WebGoat + WebWolf** (detailed tutorials with vulnerable apps and exploitation).
- **Software Assurance Maturity Model (SAMM)**: provides an effective and measurable way to analyze and improve secure development lifecycle
- **DevSecOps Maturity Model:** shows security measures which are applied when using DevOps strategies and how these can be prioritized
- **Web/Mobile Security Testing Guide (WSTG/MSTG):** testing resources for web/mobile application developers and security professionals.
- **Applications Security Verification Standard (MASVS):** provides a basis for testing web application technical security controls and also provides developers with a list of requirements for secure development.

Cyber Security Standardization Projects

- **Common Vulnerabilities and Exposures (CVE):** provides a list of common identifiers for publicly known cybersecurity vulnerabilities, used by individuals and within products to enhance security and enable automated data exchange.
- **Common Weakness Enumeration (CWE):** community-developed list of common software and hardware security weaknesses that serves as a common language, a measuring stick for security tools, and as a baseline for weakness identification, mitigation, and prevention efforts.
- **Common Attack Pattern Enumeration and Classification (CAPEC):** comprehensive dictionary and classification taxonomy of known attacks that can be used by analysts, developers, testers, and educators to advance community understanding and enhance defenses.

CWEs mentioned throughout the course

[CWE-79] Cross-site scripting (XSS)

- A special type of injection attacks, in which malicious scripts are injected into otherwise benign and trusted websites.
- **Reflected XSS:** occurs when user input is reflected in the browser without validating that it can be safely rendered, and without permanently storing the user provided data.
- **Stored XSS:** occurs when user input is stored in the website without validating that it can be safely rendered, more serious than reflected XSS as it allows attacker payload to execute in the browsers of normal website visitors (e.g., the payload can be stealing user cookies and submitting them to an attacker-controlled domain).
- **DOM-based XSS:** occurs when JS takes data from an attacker-controllable source, such as the URL, and passes it to a sink that supports dynamic code execution, such as `eval()` or `innerHTML`. This enables attackers to execute malicious JavaScript, which typically allows them to hijack other users' accounts

[CWE-352] Cross-Site Request Forgery (CSRF)

- CSRF attacks works by tricking victim browser into submitting an attacker-crafted (forged) request to a CSRF-vulnerable app in which the victim is currently logged in.
- **Example:**
  - User visits attacker website which automatically submits a request like `POST /delete_my_accout` with `host: vuln.com`.
  - If the user is currently authenticated in `vuln.com`, this will delete their account as the browser may include the session cookie in the request.
- **Countermeasures:**
  - The common fix for this is the **CSRF-token**, which is a random one-time-token generated by the server and is expected to be sent back along with POST requests generated by the website itself.
  - The above example won't work anymore because the attacker's hidden form won't include the CSRF-token which will make the server reject the request.
  - This mechanism is now included in the development frameworks so developers typically don't need to write this CSRF-token logic manually.
  - Other countermeasures include:
    - The usage of **SameSite attribute** on session cookies, which tells the browser to only send the session cookie when the request is coming from the same site that provided the cookie.
    - The usage and validation on other headers such as host, origin, or referer.
    - Require re-authentication and/or CAPTCHA and/or one-time-tokens for state-changing operations (e.g., changing password, deleting account, etc.)
    - Not using GET requests for state-changing operations.

[CWE-434] Unrestricted Upload of File with Dangerous Type

- The software allows the attacker to upload or transfer files of dangerous types that can be automatically processed within the product's environment.

[CWE-601] Open Redirect

- Occurs when an application incorporates user-controllable data into the target of a redirection in an unsafe way. An attacker can construct a URL within the application that causes a redirection to an arbitrary external domain.

[CWE-113] HTTP Response splitting

- Occurs when attacker manipulates the server into sending CRLF sequence (the official separator between HTTP headers) followed by attacker-crafted headers into the response, allowing the attacker to set arbitrary headers, change body, or split the response into two or more responses.

[CWE-840] Business-logic vulnerabilities

- Flaws in the design and implementation of an application that allow an attacker to elicit unintended behavior.
- **Example:** attacker deletes or changes the value for parameters in request body and the server doesn't validate (e.g., deleting the commission parameter or changing the amount when buying online products).

[CWE-338] Usage of weak PRNG

- The product uses a Pseudo-Random Number Generator (PRNG) in a security context (e.g., generating session id tokens), but the PRNG's algorithm is not cryptographically strong.

[CWE-78] OS command injection:

- AKA: Improper Neutralization of Special Elements used in an OS Command
- Example: the application allows the injection of bash/powershell, or other code in payload, allowing RCE.

[CWE-287] Improper Authentication

- When an actor claims to have a given identity, the software does not prove or insufficiently proves that the claim is correct.
- **Example:** the usage of poor HTTP authentication systems that can be bypassed or brute-forced.

[CWE-548] Directory Listing

- A directory listing is inappropriately exposed, yielding potentially sensitive information to attackers.
- **Example:** the exposure of files (backups, uploads, or even source-code) on the server due to webserver misconfiguration.

[CWE-413] Improper Resource Locking

- The software does not [correctly] lock resources -when exclusive access is required.
- **Technical impact:** modification of application data, DoS (instability, crash, exit, or restart).

[CWE-355] User-interface security issues:

- General issues related or introduced in the UI.
- **Example:** missing masking of sensitive data (e.g., passwords, card numbers, etc).

[CWE-639] Insecure Direct Object References (IDOR)

- **AKA:** Authorization Bypass Through User-Controlled Key.
- The system's authorization functionality does not prevent one user from gaining access to another user's data or record by modifying the key value identifying the data. (e.g., `http://vuln.com/doc/1234`).

[CWE-22] Path Traversal

- A special type of IDOR that allows accessing internal server files (outside the intended directories).
- **Example**: `http://vuln.com/../../../../etc/passwd` exposes `/etc/passwd` data due to webserver misconfiguration.

[CWE-306] Missing Authentication for Critical Function:

- The software does not perform any authentication for functionality that requires a provable user identity or consumes a significant amount of resources.

[CWE-532] Insertion of Sensitive Information into Log File:

- Information written to log files can be of a sensitive nature and give valuable guidance to an attacker or expose sensitive user information.

[CWE-943] Improper Neutralization of Special Elements in Data Query Logic

- The application generates a query intended to access or manipulate data in a data store such as a database, but it does not neutralize or incorrectly neutralizes special elements that can modify the intended logic of the query.
- **Example:** Injections of data to NoSQL databases through the use of logical operations, regex, or adding new key=value records.

[CWE-1321] Prototype pollution:

- The software receives input from an upstream component that specifies attributes that are to be initialized or updated in an object, but it does not properly control modifications of attributes of the object prototype.
- **Example:**
  - Injecting properties into existing JS object prototypes to compromise applications in various ways.
  - Example payload: `__proto__[innerHTML][]=javascript:alert(document.domain)` that will show an alert whenever `innerHTML` property is accessed from any object.

Penetration Testing

- Penetration testing is a security exercise where a cyber-security expert attempts to find and exploit vulnerabilities in a computer system.
- **Goals:** identifying system weaknesses, testing security process of the organization, verification of security compliance, finding 0-day exploits before someone else does, or bug bounty (for rewards).
- **Types:**

    - **Blackbox testing:** pentester has no information about the system, only an entry point and scope.
    - **Whitebox testing:** pentester has complete knowledge of the system (documentation, source code, architecture).
    - **Greybox testing:** pentester has partial knowledge about the system (e.g., documentation of internal data structures, algorithms used).
- **Sociotechnical penetration testing** is the process of finding the weakest users in order to gain access to confidential information.
- **Typical steps:**
    - **Planning and reconnaissance:** gathering intelligence about the organization and potential exploitation targets, defining test goals.
    - **Scanning / Information gathering:** using scanning tools to understand how a target responds to intrusions.
    - **Gaining access:** infiltrate the infrastructure by exploiting security weaknesses, then trying to escalate privileges.
    - **Maintaining access**: imitate an advanced persistent threat (e.g., install backdoors, setup services with autorun, setup covert channels, create additional accounts) to see if the vulnerability can be used to maintain access.
    - **Analysis and Reporting:** preparing a detailed report about the entire process and findings and recommendations for mitigation, the process may then repeat.
- **Open Source Intelligence (OSINT):**
    - Set of methods used to search, collect and analyze information obtained from publicly available sources.
    - **Common intelligence gathering techniques:** search engine queries, domain name searches (whois), social engineering, tax records, internet footprinting, internal footprinting, dumpster diving, tailgating.

Software Security

Recall

- Machine code, assembly, instruction set architecture, CPU registers (flags register, segment registers).
- Memory layout of a program: kernel, stack, heap, data, text.
- Program control flow, shared vs static libraries, static vs dynamic linking.
- Process Linkage Table (PLT) and Global Offset Table (GOT).
- System calls.

Binary exploitation (pwn)

- Attacks on binary files aimed at changing or taking control over the program execution flow (e.g., by redirecting jumps to other memory locations), substituting system calls, bypassing checks, substituting addresses of called functions, etc.
- To find vulnerabilities in binaries, we typically use reverse engineering: the process of taking a piece of software or hardware and analyzing its functions and information flow so that its functionality and behavior can be understood.
- **Buffer overflow:**
    - An anomaly that occurs when the size of data exceeds the storage capacity of the memory buffer.

- As a result, the program attempting to write the data to the buffer overwrites adjacent memory locations.
- This can be exploited in many ways to inject and execute malicious code, expose or overwrite data.
- Unsafe C library functions and their safe equivalents:

Unsafe function	Safe equivalent	Purpose
gets()	fgets()	Read characters
strcpy()	strncpy()	copy string to a buffer
strcat()	strncat()	buffer concatenation
sprintf()	snprintf()	fill buffer wuth data
scanf("%s", buf)	scanf("%9s", buf)	Read user input into `char buf[10];`

Mobile Application Security

Recall

- **Android architecture**
  - System apps, Java API Framework, Native C/C++ libraries, Android Runtime (ART), Hardware Abstraction Layer (HAL), Linux Kernel.
  - View System, Resource Manager, Notification Manager, Activity Manager, Content Providers.
  - Java Virtual Machine (JVM), Dalvik Virtual Machine (DVM), Android Runtime (ART). [check] [check].
  - Ahead-of-Time (AOT) and Just in Time (JIT) compilation.
  - D8 and R8.
- **iOS Architecture**
  - Cocoa Touch, Media Layer, Core Services, Core OS, XNU Kernel and Device Drivers.
  - SpringBoard, Sandbox
- **Mobile Applications**
  - Native vs Hybrid Mobile Apps
  - Xamarin (Mono) for Android and iOS.
- **Permission system in Android and iOS**
  - User-installed applications have limited access to the system, resources, and data.
  - User must explicitly grant permission to an application to use its features.
  - Application are typically forced to implement certain security mechanisms (e.g., HTTPS), and the OS is the one taking care of encryption, storage, etc.
  - **Android:** different permissions categories: install-time, run-time, and special permissions.
  - **iOS:** apps must request permissions to be able to access personal data, user-generated content, protected resources, or device capabilities.
- **Root and Jailbreak**
  - Bypassing OS security mechanisms by getting a non-standard access to the system or parts of it (e.g., a privileged user).

OWASP Mobile Top 10 (2016)

A classifier of key security risks in mobile applications, ranked by popularity and criticality.

1. Improper Platform Usage:

- **Description:** developer errors; accidental or intentional misuse of platform security mechanisms
- **Examples:** exposing application data to other applications, storage of authentication information in an open form or in a storage accessible to all, misuse of biometrics.
- **Countermeasures:** following vendor-provided guidelines.

2. Insecure Data Storage

- **Description:** attacker was able to steal user data by having physical access to the device or tricking user into installing a malicious app.
- **Example:** the usage of unencrypted data storage for sensitive data.
- **Countermeasures:** try to reduce the amount of sensitive data used by the application, or make sure it's secured properly.

3. Insecure Communications

- **Description:** attacker was able to eavesdrop on device communications.
- **Examples:** malware installed on a device can read and forward outgoing packets. Application uses weak or doesn't use encryption for transmitted data. User is connected to a compromised network.
- **Countermeasures:** application-level encryption, verification of certificates.

4. Insecure Authentication

- **Description:** attacker was able to send a valid request without being authenticated into the application.
- **Example:** traffic analysis or application code analysis allowed the attacker to craft a request and use the application without being authenticated.
- **Countermeasures:** proper design/implementation/ testing of authentication systems, usage of secure access tokens with the ability to revoke them in case device is lost/stolen.

5. Insufficient Cryptography

- **Description:** attacker was able to perform obtain authentication data of the user or perform actions on behalf of them because of incorrect usage of cryptography.
- **Example:** offline app with password authentication is storing unsalted password hash in MD5 in a local database.
- **Countermeasures:** avoid storing sensitive data on the device, follow recommendations when using cryptography.

6. Insecure Authorization

- **Description:** attacker was authorized do actions on a system they were not supposed to have access to.
- **Example:** usage of automation tools to get privileged access to application backend API.
- **Countermeasures:** requests to the backend must be checked for authorization.

7. Client Code Quality

- **Description:** vulnerabilities related to poor quality code.
- **Example:** SQL injection on android SQLite.
- **Countermeasures:** pentesting, code reviews, QA.

8. Code Tampering

- **Description:** modifying client-side application code to gain higher privileges or do malicious activities.
- **Example:** the usage and distribution of cracked client-side apps.
- **Countermeasures:** root/jailbreak/debugging tools detection, integrity checks for code.

9. Reverse Engineering

- **Description:** attacker found and exploited vulnerabilities by reverse engineering application code.
- **Example:** usage of Java decompiler and similar tools to get information about backend addresses and check application logic for flaws.
- **Security measures:** reverse engineering can not be prevented, but can be made harder through code obfuscation and similar mechanisms.

10. Extraneous Functionality

- **Description:** attacker exploited a hidden functionality on the backend.
- **Example:** brute-forcing hidden API endpoints that is used for development or some unreleased/unused feature.
- **Countermeasures:** protect or remove redundant or unused functionality.

Network Security

Recall

- Network Segmentation
- VLANs: port-based vs tagged.
- Software Defined Networking (SDN)
- Service mesh, microservices.
- Firewalls, Next Generation Firewalls (NGFW)
- IDS/IPS, WAF, DLP, VPN.