OWASP Top 10 Cheat Sheet

1. Broken Access Control

Users having the ability to perform *unauthorized* actions, such as

- Reading, modifying, or deleting information
- Performing business functions

Frequency: More occurrences than any other category.

Common Vulnerabilities

- Missing access controls
- Insecure Direct Object References (IDOR)
- HTTP verb tampering
- JWT or cookie tampering
- CORS allowing API access to untrusted origins

Best Practices

- Proper Access Control
  - Access control in **server-side code!**
  - **Deny** by default
  - Principle of **least** privilege
  - **Reuse** access control mechanisms
  - Enforce record **ownership**
  - **Invalidate** sessions after logout
- Configuration
  - **Minimize** CORS usage
  - **Disable** directory listings
  - Keep file metadata (i.e., .git) and backup files **outside of web root**
- **Log** access failures with alerts upon repeated failures
- Apply API **rate limiting** to mitigate automated attacks

2. Cryptographic Failures

Data protection in transit and at rest.

Best Practices

- **Identify** sensitive/protected data
- Apply required controls
- **Discard** sensitive data as soon as possible
- **Encrypt** sensitive data at rest
- Use **up-to-date** algorithms, protocols, and keys
- Use **proper** key management
- Encrypt **in transit** and enforce it (HTTP Strict Transport Security)
- **Disable caching** of sensitive data
- Store passwords using **strong** adaptive and **salted** hashing functions
- Use a **Cryptographically secure** pseudo random number generator for Initialization Vectors (IVs)
- Use an Initialization Vector (IV) only **once**
- Use Authenticated Encryption (i.e. **HTTPS**)
- Ensure **good seed** is used for cryptographic randomness. Modern APIs do not require developers to seed the CSPRNG.
- **Avoid deprecated** functions (i.e. MD5, SHA1, PKCS number 1 v1.5)
- Independently verify effectiveness

3. Injection

When *untrusted* user input is passed and *parsed* by an interpreter.

Attack varies by interpreter.

Applies to

- SQL
- NoSQL
- OS commands
- HTTP/JavaScript (XSS)
- XML External Entity (XXE)
- And more

Best Practices

- **Don't trust** user input
- Use a **parameterized** interface
- **Sanitization**
  - Prefer server-side "**allow list**" validation (define what IS allowed)
  - Otherwise, use deny list validation or escape special characters specific to the interpreter(s)
- SQL `LIMIT` can help mitigate disclosure when SQL injection is successful

4. Insecure Design

Broad category for "missing or ineffective control design" with ***design*** being the key focus.

Also see the
OWASP Software Assurance Maturity Model (SAMM).

Common Weakness Enumerations

- CWE-209: Generation of Error Message Containing Sensitive Information
- CWE-256: Unprotected Storage of Credentials
- CWE-501: Trust Boundary Violation
- CWE-522: Insufficiently Protected Credentials

Best Practices

- Define security **requirements** (confidentiality, integrity, availability)
- Establish a **Secure Development Lifecycle**
- Utilize **secure design patterns** and **reference architectures**
- Integrate **Threat Modeling** into refinement sessions (or similar sessions)
  - Look for changes in **data flows**, **access control**, or other security controls
  - Document results
  - Learn
  - Offer **positive** incentives
- Incorporate security in **user stories**
  - Create **automated** security tests
  - **Test** that all critical flows are resistant to the threat model
- **Segregate** tenants robustly
- **Limit** resource consumption

5. Security Misconfiguration

Any gaps in management (configuration, updates, lack of removal) of software leading to vulnerabilities.

Common Issues

- Lack of security hardening
- Improperly configured permissions on cloud services
- Unnecessary features enabled/installed
- Default accounts
- Verbose error handling
- Out-of-date software

Best Practices

- Dev/Prod parity
  - All environments (Dev, QA, Prod) should be as **identical** as possible
- **Remove unused** software / features
- As part of the **patch** management process, **review**/update any configurations based on security notes/updates.
- Employ a **segmented** application architecture
  - Cloud security groups (**ACLs**)
- Use **HTTP security headers**
- Use an **automated** process to verify effectiveness and settings in all environments

6. Vulnerable and Outdated Components

Avoiding vulnerabilities through dependencies.

You Are Likely Vulnerable If

- If you **do not know** the versions of all components you directly use (including nested dependencies)
- If any software is **vulnerable**, **unsupported**, or **out-of-date**
- If you **do not scan** for vulnerabilities **regularly** and **subscribe** to security bulletins related to the components you use
- If you don't fix or upgrade in a **timely fashion**. (aka. monthly/quarterly patching)
- If you do not **secure** the components' configurations

Best Practices

- **Remove unused** dependencies and software
- Continuously **inventory** versions of software
- **Subscribe** to security bulletins for components in use
- Prefer **signed** packages from **official** sources
- **Monitor** for components that are unmaintained

7. Identification and Authentication Failures

Confirming user identities, authentication, and session management.

Common Weaknesses

- Allow **automated** attacks (i.e., credential stuffing)
- Allow **brute force** attacks
- Allow **weak passwords**
- Use **weak credential recovery** like question-answer challenges
- **Plaintext**, **encrypted**, or **weak** hash password stores
- **Missing 2FA**
- **Exposing session** identifier in URL
- Session identifier **reuse**
- Failing to **invalidate** session IDs (logout, etc.)

Best Practices

- Where possible, **implement 2FA**
- Do not deploy any **default** accounts
- **Check passwords** against the top 10,000 worst passwords list
- Good password **length**, **complexity**, **rotation**
- Good account/credential messages to avoid account enumeration
  - Give the same message for successful and unsuccessful attempts
- **Increasingly delay** login on multiple failed attempts
- **Alert** admins on multiple failed logon attempts
- Good **session management**

8. Software and Data Integrity Failures

Ensuring app, plugins, etc. come from trusted sources.

One of the **highest** weighted impacts.

Without sufficient integrity checks, ingested data or updates could result in malicious code execution.

Best Practices

- Use **digital signatures** to verify software/data
- Use **trusted** repositories. Consider hosting an internal vetted repository.
- Use a **supply chain security tool** to ensure dependencies do not have vulnerabilities.
- Have a **review process** for code and configuration changes to mitigate introduction of malware
- **Protect build and deploy processes** within the CI/CD pipeline with proper segregation, configuration, and access control
- Perform **integrity checks** on unsigned or unencrypted serialized data

9. Security Logging and Monitoring Failures

Ensuring that attacks can be detected and responded to.

The following transactions should be logged:

- Logins
- When rate limits are exceeded
- High-value transactions
- Input validation failures

Best Practices

- Ensure that logs have the following
  - **Clear** message
  - Sufficient **user context**
  - Stored **long enough** to analyze malicious activity
  - Easily **ingestible** for log management solutions
  - Encoded correctly to **prevent injections**
- Automated **alerts** of suspicious activity
- Quick **response** to alerts
- Utilize an **incident response and recovery plan**

10. Server Side Request Forgery (SSRF)

When *untrusted* user input becomes part of *fetching* remote resources and causes the server-side application to make requests to an *unintended* location.

Potential Issues from SSRF

- Exploiting **trust** relationships to access other internal systems (revealing info)
- Connecting to external systems (potentially leaking auth credentials)
- Could result in arbitrary command execution

Best Practices

- Network Defenses
  - **Separate** networks and services (treat the server as "untrusted" to mitigate impact)
  - "**Deny by default**" firewall or network access control rules
  - Establish ownership and lifecycle for firewall rules
  - **Log** all accepted and blocked network flows on firewalls
- Application Layer Defenses
  - **Sanitize** and **validate** client data on **server**
  - Use a **Positive Allow** list
  - **Do not send raw responses** to clients
  - Disable HTTP redirections