OWASP TOP 5 WEB APPLICATION SECURITY RISKS (2025)

An Overview of Major Web Vulnerabilities

By, Ameen ul Islam
MES College, Marampally

CONTENTS

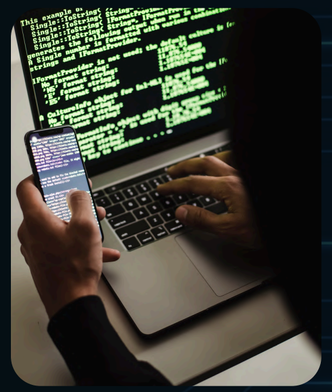
INTRODUCTION TO OWASP

What is OWASP?

- Open Web Applications Security Project
- A global open community focused on improving application & API security
- Provides free tools, standards, research, cheat sheets, and learning resources

Why OWASP Top 5?

- Highlights the most critical, common application security risks using large-scale data
- Helps defend against attacks exploiting many paths through an application
- Prevents real-world impacts like data exposure, system compromise, and business losses



New Challenges in the Digital World

PROBLEM STATEMENT

The Problem

- Modern applications expose many different paths attackers can exploit
- Misconfigurations, weak controls, and outdated components are commonly found across systems
- Developers often repeat the same vulnerabilities due to lack of awareness and secure coding practices

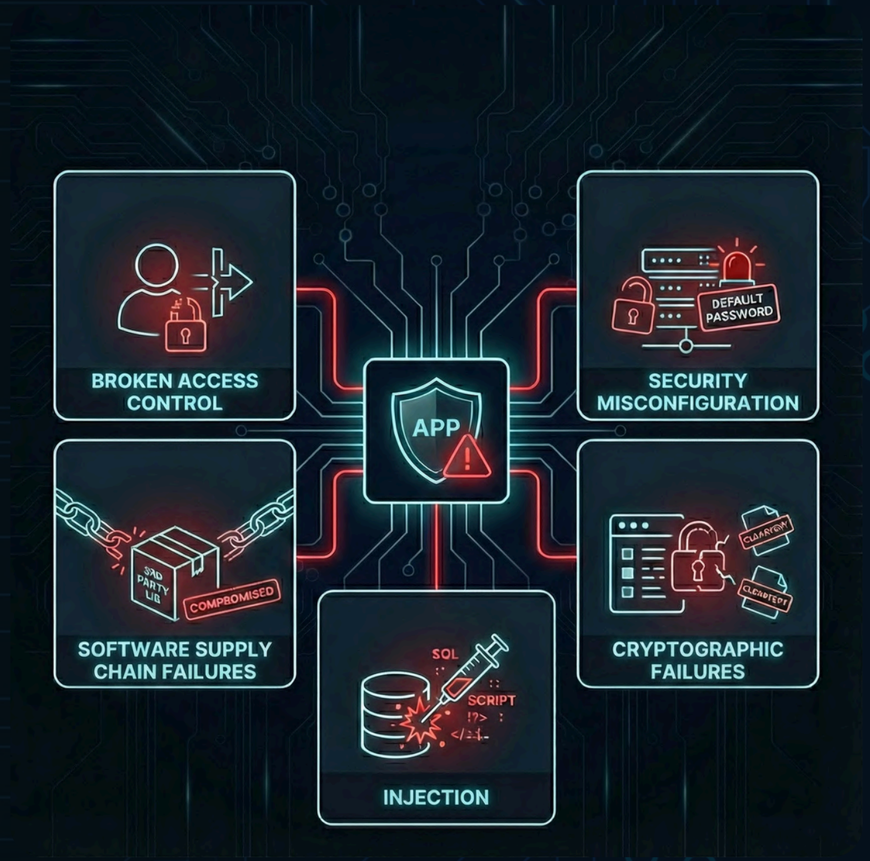
Why this Matters

- A single missed control can lead to unauthorized access, data exposure, or full system compromise
- Increased software complexity makes security harder to manage and easier to break
- Organizations face real business impact when technical vulnerabilities are exploited

OBJECTIVES

Objectives of this Seminar

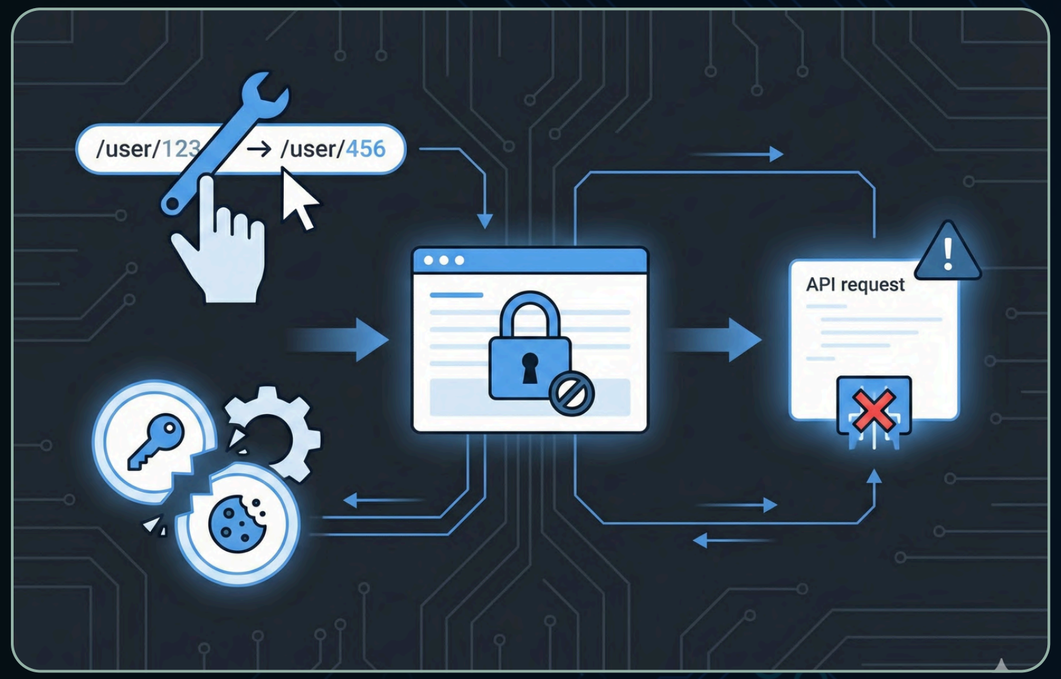
- To familiarize the audience with the OWASP Top 5 vulnerabilities
- To understand how each vulnerability occurs and how to detect/match it
- To present real-world examples and statistics to show practical impact
- To highlight recommended prevention and mitigation practices
- To emphasize why addressing these risks strengthens overall application security



OVERVIEW OF OWASP TOP 5 (2025)

Top 5 Application Security Risks

- Broken Access Control — Users can act outside intended permissions
- Security Misconfiguration — Incorrect or insecure system settings
- Software Supply Chain Failures — Compromise through third-party components
- Cryptographic Failures — Weak or missing encryption and key management
- Injection — Untrusted input executed as commands or queries



BROKEN ACCESS CONTROL

What it is

- Failures that allow users to act outside their intended permissions (view, modify, delete data)
- Leads to unauthorized access, information disclosure, or business logic abuse
- A fundamental failure that directly breaks application trust boundaries

How it Happens

- Access rules are incomplete, inconsistent, or enforced only on the client side
- Missing or weak server-side checks, enabling URL/API tampering
- Insecure Direct Object References (IDOR) exposing other users' data
- Privilege escalation by modifying cookies, JWTs, hidden fields, or metadata

DETECTION & MATCHING

How Attackers Detect It

- Try modifying IDs, parameters, or URLs to access other users' data (IDOR)
- Attempt force browsing to restricted or admin-only endpoints
- Replay or tamper with JWTs, cookies, or session tokens to escalate privileges

How Developers/Testers Match It

- Check if API endpoints enforce server-side authorization consistently
- Test POST/PUT/DELETE APIs for missing access controls
- Inspect CORS configuration for acceptance of unauthorized origins
- Include access control checks in unit and integration tests

A01: Broken Access Control

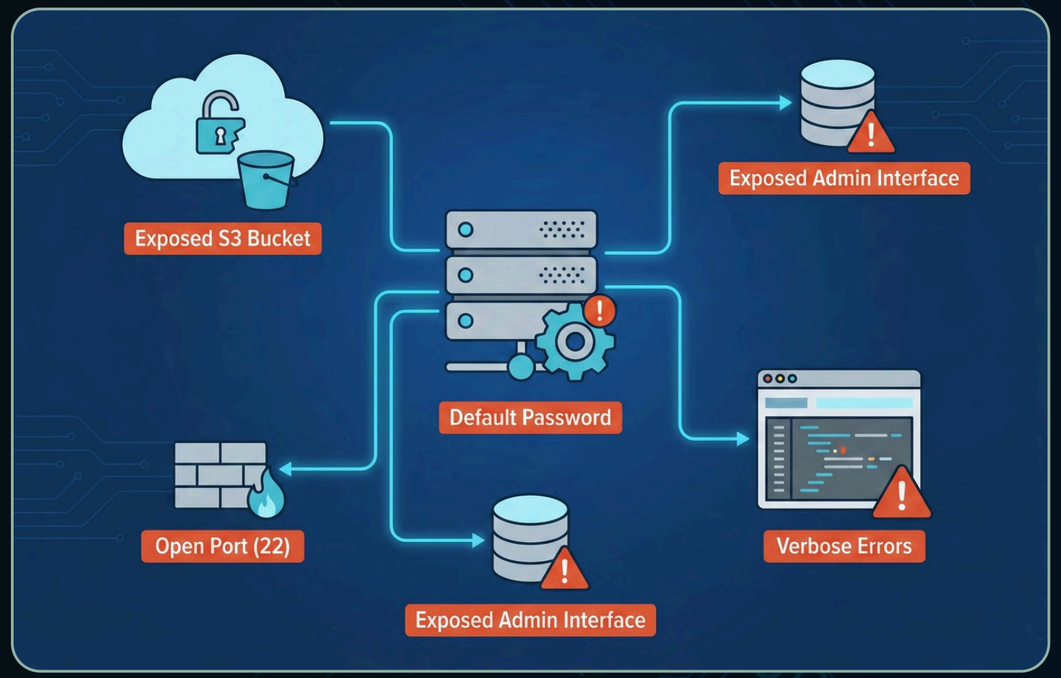
STATISTICS & REAL WORLD EXAMPLE

Statistics

- 40 CWEs mapped under this category — the highest among all risks
- 1.8M+ occurrences reported in contributed datasets
- 3.74% average incidence rate across applications; 20.15% max incidence rate
- 32,654+ CVEs associated — second highest CVE count overall

Real-World Incident: Optus 2022 Data Breach

- In 2022, Optus disclosed a breach that exposed personal data of nearly 10 million current and former customers.
- Regulatory filings identified a coding error in access control on a dormant API — effectively allowing unauthorized requests to reach sensitive customer records.
- The flaw went undetected for years, demonstrating how a single missing server-side authorization check can compromise massive amounts of data.



SECURITY MISCONFIGURATION

What it is

- Incorrect or insecure settings across the application stack, cloud services, or servers
- Occurs when default configurations, unused features, or verbose error messages are left enabled
- Often overlooked during deployment and maintenance, making it a high security risk

How it Happens

- Missing hardening: unnecessary ports, services, accounts, or default credentials left active
- Outdated or insecure configuration values in servers, frameworks, libraries, or cloud storage
- Detailed error messages reveal stack traces or system information to attackers

DETECTION & MATCHING

How Attackers Detect It

- Probe for default credentials, sample apps, or leftover admin interfaces
- Check for open directories, unnecessary services, or exposed cloud buckets
- Trigger errors to extract stack traces, version info, or internal paths

How Developers/Testers Match It

- Review configurations across servers, frameworks, cloud storage, and containers for deviations from secure defaults
- Confirm unused features, sample apps, and debug modes are disabled
- Use automated tools to audit settings: misconfigured permissions, weak CSP headers, or overly permissive ACLs

A02: Security Misconfiguration

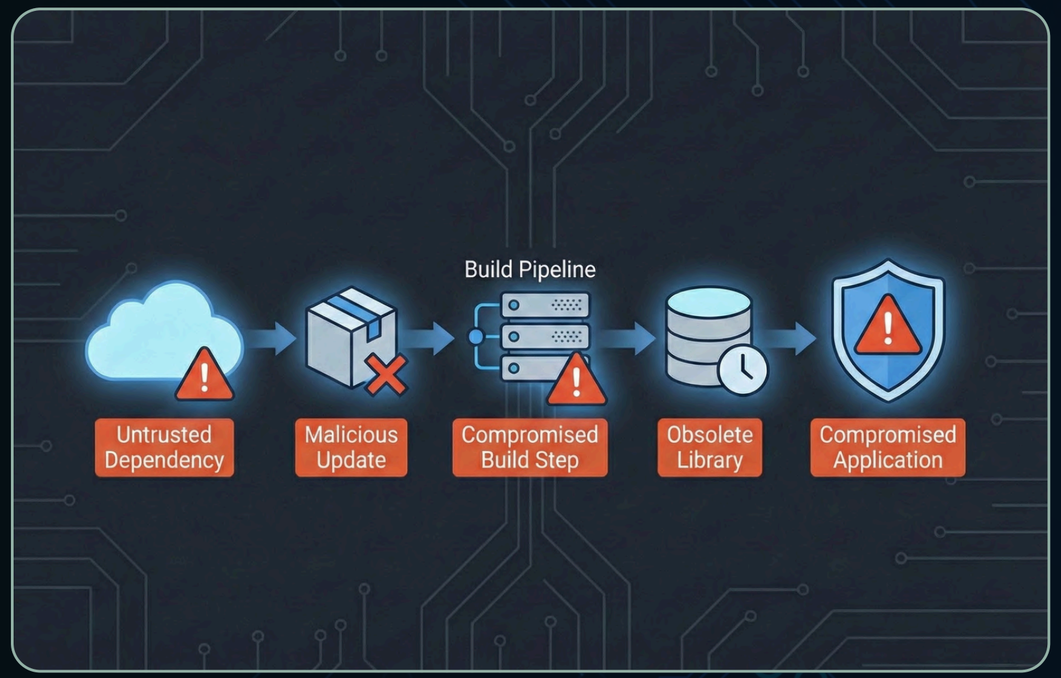
STATISTICS & REAL WORLD EXAMPLE

Statistics

- Commonly observed in real-world application assessments
- 16 CWEs mapped under this category
- 3.00% average incidence rate; 27.70% max incidence rate across tested apps
- Over 719,084 occurrences reported in contributed data

Real-World Incident: Capital One Cloud Misconfiguration (2019)

- A misconfigured AWS S3 bucket + overly permissive IAM role allowed an attacker to access sensitive customer data.
- Over 100 million customer records were exposed, including names, addresses, credit scores, and social security numbers.
- Root cause: A WAF (Web Application Firewall) role was incorrectly configured, enabling SSRF-based access to internal cloud resources.



SOFTWARE SUPPLY CHAIN FAILURES

What it is

- Compromise or weaknesses in third-party libraries, tools, or build pipelines an application depends on
- Includes outdated, vulnerable, or malicious components used during development or deployment
- Often hard to detect because issues originate outside the main application code

How it Happens

- Using outdated, unmaintained, or vulnerable libraries and transitive dependencies
- Lack of tracking, auditing, or verifying changes in CI/CD pipelines, repositories, or IDE tooling
- Downloading components from untrusted sources or without verifying package integrity/signatures

DETECTION & MATCHING

How Attackers Detect It

- Scan for known vulnerable libraries used in apps or APIs
- Target open-source projects with low activity or weak maintainer verification
- Poison public package repositories with similarly named malicious packages (e.g., typo-squatting)

How Developers/Testers Match It

- Use SBOMs (Software Bill of Materials) to list all components and their sources
- Run dependency vulnerability scanners (e.g., Snyk, Dependabot, osv-scanner)
- Validate package integrity using cryptographic signatures or checksums before build/deploy

A03: Software Supply Chain Failures

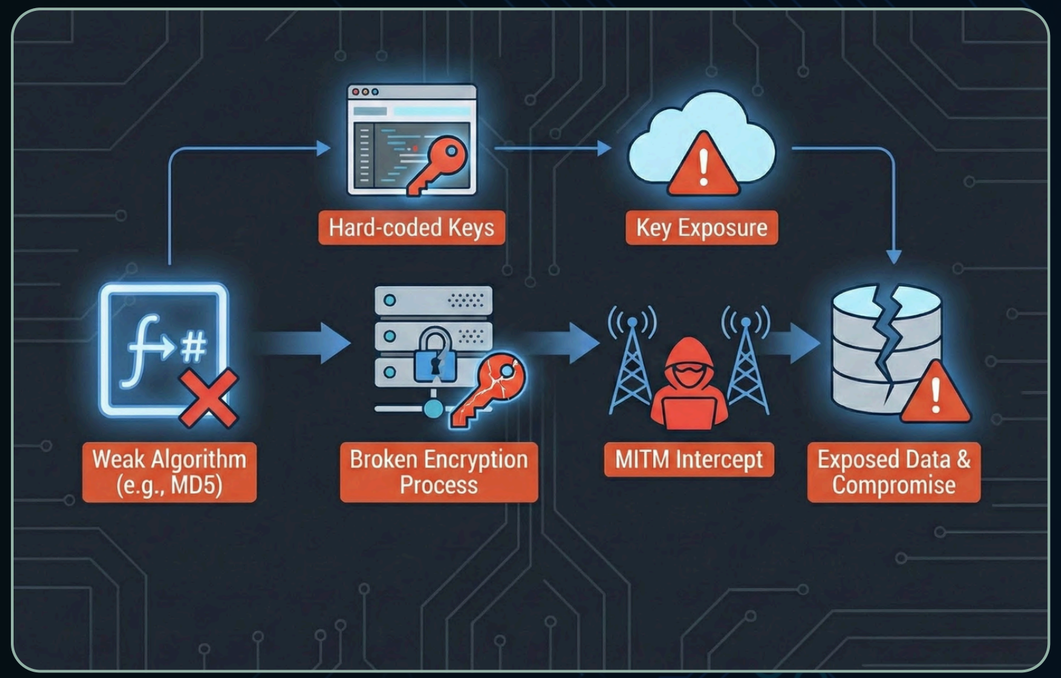
STATISTICS & REAL WORLD EXAMPLE

Statistics

- 5 CWEs mapped to this category
- Highest average incidence rate: 5.19% among all tested categories
- Over 215,248 occurrences reported in OWASP data contribution
- Only 11 CVEs directly mapped — showing how difficult these failures are to detect and classify

Real-World Incident: SolarWinds Orion Attack (2020)

- Attackers tampered with the SolarWinds build pipeline, injecting malicious code into Orion software updates
- Compromised updates reached ~18,000 organizations, including government agencies and Fortune 500 companies
- Considered one of the most impactful supply-chain breaches in recent history



CRYPTOGRAPHIC FAILURES

What it is

- Weak, missing, or improperly implemented cryptographic protections for sensitive data
- Includes use of outdated algorithms, weak randomness, or poor key management
- Often leads to exposure of passwords, personal data, or confidential information

How it Happens

- Use of deprecated or insecure algorithms (e.g., MD5, SHA-1, weak PRNGs)
- Poor key handling: default keys, reused keys, or storing keys in code repositories
- Failure to enforce encryption in transit or at rest (missing TLS, plaintext storage)
- Improper use of IVs/nonces — reused or predictable values weaken encryption modes (e.g., ECB, CBC)

DETECTION & MATCHING

How Attackers Detect It

- Inspect traffic for missing TLS, weak ciphers, or downgraded connections (MITM opportunities)
- Look for exposed or hard-coded cryptographic keys in repositories or client-side code
- Analyze application responses for clues like deprecated hash functions, weak randomness, or predictable IVs

How Developers/Testers Match It

- Confirm use of strong, updated algorithms; avoid deprecated ones (MD5, SHA-1, ECB)
- Review key management: rotation, secure storage, no keys in source code
- Test TLS settings, certificate validation, and mandatory encryption for sensitive data

A04: Cryptographic Failures

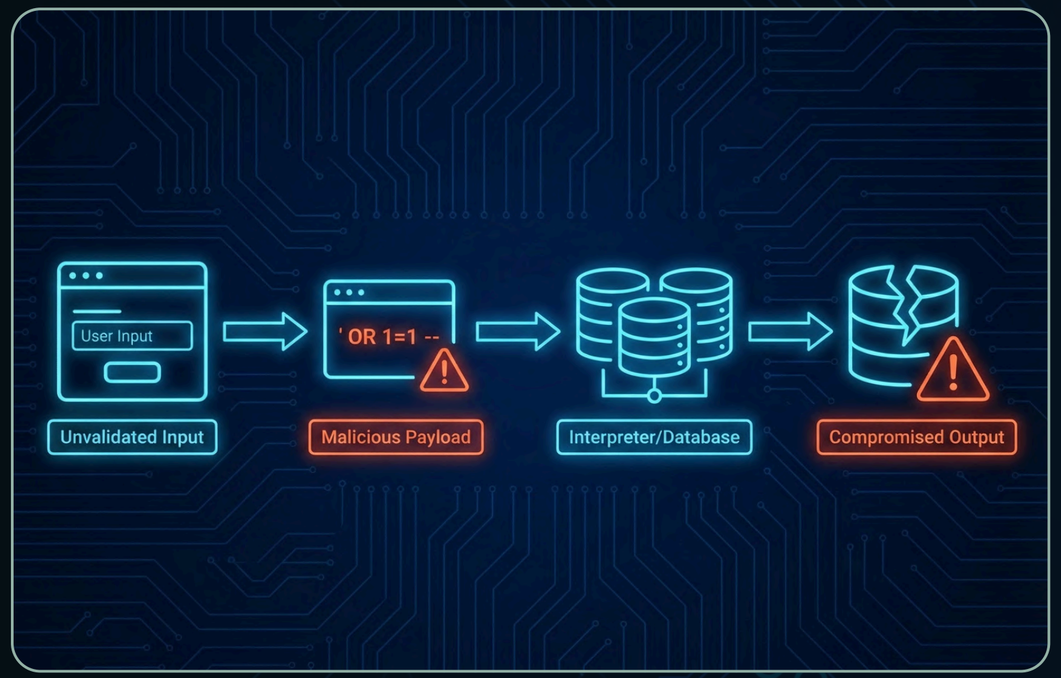
STATISTICS & REAL WORLD EXAMPLE

Statistics

- 32 CWEs mapped under this category
- 3.80% average incidence rate; 13.77% max incidence rate across tested apps
- Over 1.6 million occurrences reported in OWASP data contributions
- More than 2,185 CVEs associated with cryptographic weaknesses

Real-World Incident: WhatsApp Encryption Downgrade Flaw (2023)

- A flaw in WhatsApp’s encryption pipeline allowed certain message sessions to be downgraded to weaker cryptographic states during edge-case transitions
- Attackers with man-in-the-middle (MITM) capabilities could exploit improper key verification logic
- Demonstrated that even strong end-to-end encryption can fail if crypto implementation and validation logic are misconfigured



INJECTION

What it is

- When untrusted input is interpreted as code or commands by the application
- Allows attackers to alter queries, execute commands, or access unintended data
- Includes SQL, NoSQL, OS command, LDAP, ORM, and template injections

How it Happens

- User input not validated, sanitized, or filtered before reaching interpreters
- Dynamic queries or non-parameterized statements built directly from user data
- Frameworks or ORMs blindly trusting user-controlled fields (e.g., HQL injection)

DETECTION & MATCHING

How Attackers Detect It

- Inject special characters or payloads (', ", --, ${}, {{ }}) to observe changes in response behavior
- Test parameters, headers, cookies, and JSON/XML bodies for interpreter errors or delays (e.g., SLEEP(10))
- Use automated fuzzers or scanners to identify unvalidated inputs reaching backend interpreters

How Developers/Testers Match It

- Review code for dynamic queries, concatenated strings, or any interpreter calls built from user input
- Use SAST/DAST/IAST tools in the CI/CD pipeline to flag unsafe input flows
- Test all input points with parameterized queries and strict server-side validation policies

STATISTICS & REAL WORLD EXAMPLE

Statistics

- 37 CWEs mapped — one of the broadest OWASP categories
- 3.08% average incidence rate; 13.77% max incidence rate across tested apps
- Over 1.4 million occurrences recorded in contributed OWASP data
- Highest CVE count: 62,445+ CVEs, dominated by XSS & SQLi

Real-World Incident: British Airways Breach (2018)

- Attackers injected malicious JavaScript into BA's website via a compromised third-party script
- The skimming script captured payment details of over 380,000 customers during checkout
- Demonstrated how injection + supply chain flaws can directly lead to large-scale data theft

ADVANTAGES & LIMITATIONS

Avantages

- Provides a clear, data-driven view of the most critical web application risks
- Widely adopted industry standard for training, audits, and secure development
- Helps developers prioritize fixes with maximum security impact

Limitations

- High-level: does not cover all vulnerabilities or provide implementation details
- Updates every few years, so emerging risks may not appear immediately
- Can be misused as a checklist rather than part of a full security program

APPLICATIONS OF OWASP

Secure SDLC

Helps integrate security into design, coding, and testing by focusing on the most critical risks.

Penetration Testing

Provides a standard baseline for creating test cases and identifying common high-impact vulnerabilities.

Compliance & Governance

Aligns with many industry and regulatory requirements, supporting stronger security governance.

Developer Training

Acts as a clear learning guide to teach teams the most frequent and dangerous security issues.

CONCLUSION

Key Points to Remember

- Always validate, sanitize, and enforce strict controls on all inputs and actions.
- Misconfigurations and outdated components are among the biggest real-world causes of breaches.
- Strong cryptography and proper key management are mandatory, not optional.
- Security must be continuous — integrated into development, testing, and deployment.

Summary

- The OWASP Top 5 highlights the most critical and frequent web application security risks.
- Each vulnerability can be exploited through predictable weaknesses in design, configuration, or input handling.
- Understanding how they occur, how to detect them, and how to prevent them is essential for secure development.

THANK YOU