Fundamentos de Segurança Informática

LEI

2025/2026

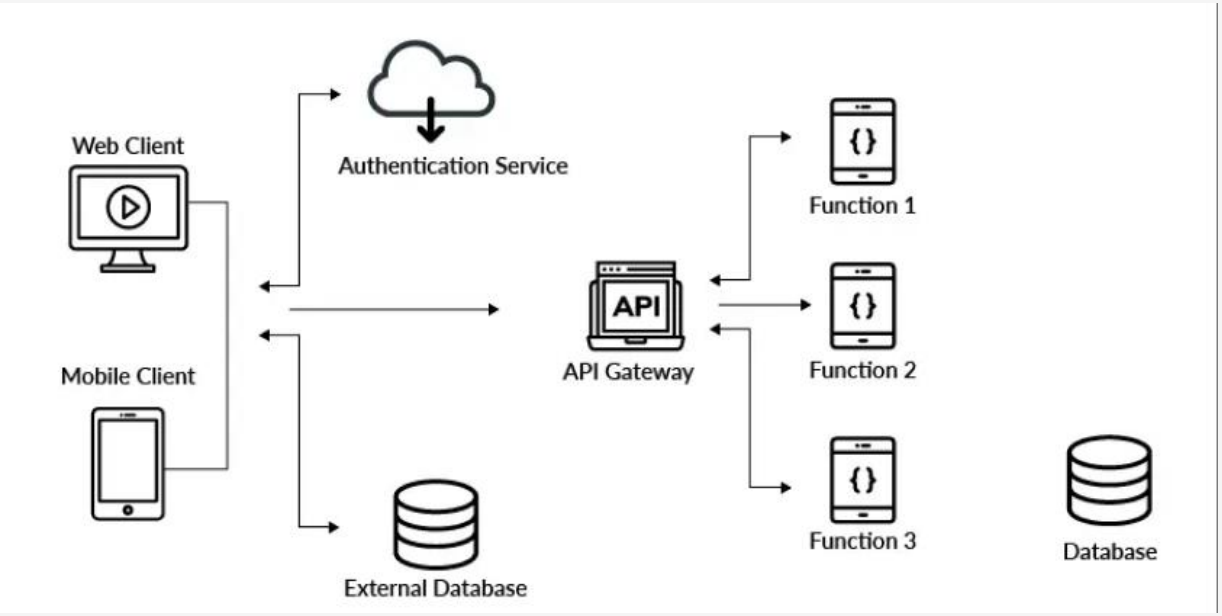
T10 – Web application security

Summary

- What is a web application
- Why are applications target of attacks?
- Attack surface of a web application
- HTTP itself is the attack vector
- OWASP project
- OWASP Top 10 risk categories
- OWASP Top 10 attacks
- WSTG
- Security Identifiers

What is a web application

- A web application is a software program that runs on a remote server and is accessed through a web browser, which acts as the client.
- Communication between the browser and the server typically uses HTTP or HTTPS over the Internet or an intranet.
- Unlike traditional desktop applications, web applications do not require installation on the client side and are accessed via a URL..



What is a web application (2)

Common examples:

Online banking, e-commerce platforms (Amazon, eBay), webmail (Gmail), social networks, SaaS tools (Google Docs, Amazon AWS), and enterprise portals.

Components:

- Front-end — HTML and CSS interpreted by the browser, JavaScript executed in the browser (client-side)
- Back-end — Application logic (Java, Python, PHP, Node.js, …) running on the server
- Database — persistent storage (MySQL, PostgreSQL, MongoDB…)
- APIs — interfaces for communication between services (REST, GraphQL)
- Web server — handles incoming requests (Nginx, Apache, IIS)

Any code that runs in the browser (HTML/CSS/JS) is not reliable for security
→ all critical validations must be performed server-side.

Why are web applications target of attacks?

Why are web application targets of attacks?

- Are publicly accessible by design, handle sensitive data (credentials, payments, personal information), and are built by large teams using many third-party components — making them a prime target.
- Broken Access Control is the most widespread vulnerability category (OWASP Top 10 2021, https://owasp.org/Top10/A01_2021-Broken_Access_Control/)
- Credential abuse (22%) and exploitation of vulnerabilities (20%) are the leading initial access vectors in confirmed breaches (Verizon DBIR 2025, https://www.verizon.com/about/news/2025-data-breach-investigations-report)

Attack surface of a web application

Entry points (inputs)

HTML Forms

Login, registration, search, checkout — every field is untrusted input

Query Strings / URL

/search?q=<payload> — visible and directly manipulable

HTTP Headers

User-Agent, Referer, Cookie, X-Forwarded-For — rarely validated

REST / GraphQL APIs

JSON/XML bodies, route parameters, introspection endpoints

File Uploads

Filenames, MIME types, content — vectors for malware/path traversal

WebSockets

Persistent bidirectional channel — bypasses stateless protections

Rules of web security

NEVER trust the client

ALL input must be validated server-side

✓ Validate type & format:

Reject on the server, not just the client — JavaScript can be bypassed

✓ Whitelist > Blacklist:

Define what is allowed; block everything else by default

✓ Encode on output:

Prevents XSS — HTML, JS, URL encode depending on context (avoids interpretation as HTML/JS by the browser)

✓ Parameterise queries:

Never concatenate input directly into SQL/LDAP/OS commands

✓ Least privilege:

Processes and accounts with only the permissions strictly required

HTTP itself is the attack vector

HTTP Request — Detailed Example

```
POST /api/login HTTP/1.1
Host: app.example.com          ← (1)
User-Agent: Mozilla/5.0...     ← (2)
Content-Type: application/json
Cookie: session=abc123         ← (3)
Authorization: Bearer eyJ...   ← (4)
X-Forwarded-For: 1.2.3.4       ← (5)
Origin: https://app.example.com

{                              ← (6)
  "username": "admin",
  "password": "secret"
}
```

Attack Vectors per HTTP field

(1) Host

Host Header Injection → password reset to attacker domain, cache poisoning

(2) User-Agent

Log4Shell — ${jndi:ldap://...} triggered RCE on vulnerable Log4j servers

(3) Cookie

Session Hijacking, CSRF, missing cookie flags (HttpOnly, Secure, SameSite)

(4) Authorization

JWT without signature verification, expired tokens accepted, alg:none

(5) X-Forwarded-For

IP-based rate limit bypass, geographic location spoofing

(6) JSON/XML Body

SQLi, NoSQLi, XXE (XML External Entities), Mass Assignment, Prototype Pollution

OWASP – Open Web Application security project

A global reference for web application security

Key OWASP Projects

OWASP Top 10

List of the 10 most critical risks in web applications. Published every 3–4 years based on real vulnerability data. Last edition from 2025.

WSTG v4.2

Web Security Testing Guide — testing guide with 91 test cases in 12 categories. Methodology used by pentesters and QA teams worldwide.

ASVS

Application Security Verification Standard — security requirements framework at 3 maturity levels. Foundation for certifications and compliance (PCI-DSS, ISO 27001).

OWASP ZAP

Zed Attack Proxy — open source vulnerability scanner. Used in CI/CD for automated detection of XSS, SQLi, CSRF and more. Free alternative to Burp Suite.

Juice Shop

Intentionally vulnerable web application for hands-on learning. Contains all Top 10 and 100+ challenges

OWASP Top 10 risk categories

A01 – Broken Access Control
Authorization flaws allow users to access data or perform actions beyond their intended permissions

A02 – Cryptographic Failures
Sensitive data is exposed due to missing, weak, or incorrectly implemented cryptography

A03 – Injection
Untrusted input is interpreted as code or commands, leading to data compromise or system execution

A04 – Insecure Design
The application lacks essential security controls due to flawed or incomplete architectural decisions

A05 – Security Misconfiguration
Insecure default settings or poor configuration expose the application to preventable attacks

A06 – Vulnerable & Outdated Components
The use of known vulnerable or unmaintained software components introduces exploitable risks

A07 – Identification & Authentication Failure
Weak authentication or session management enables account takeover and identity abuse

A08 – Software & Data Integrity Failures
The system fails to ensure the integrity of code or data, enabling supply-chain and tampering attacks

A09 – Security Logging & Monitoring Failures
Attacks remain undetected due to insufficient logging, monitoring, and alerting mechanisms

A10 – Server-Side Request Forgery
The attacker tricks the server into making requests to internal or restricted network resources

OWASP Top 10 attacks (examples)

A01 - Broken Access Control

What is it?
Broken Access Control occurs when users can access resources or actions they are not authorized to use

How it works
The server checks authentication but not authorization
Object IDs or endpoints are not protected

Typical attacks
IDOR
Privilege escalation
Access to admin endpoints

Impact
Data exposure
Account takeover
Regulatory violations

✅ **Mitigation:** Enforce server-side authorization checks on every request.

OWASP Top 10 attacks (examples)

A01 - Broken Access Control
Example: IDOR (Insecure Direct Object Reference)

The server only checks if the user is authenticated, but does not verify whether the requested resource belongs to that user. Any authenticated user can access any other user's data simply by changing a number in the URL or JSON body.

Attack Steps:

1. User authenticates and receives a valid JWT token
2. Accesses their account: GET /api/users/42/orders
3. Changes the ID to 43, 44, 45... in an automated loop
4. Server responds 200 OK with other users' data
5. Attacker extracts the full customer database (mass data leak)

Payload / Code

```
# Legitimate request (user ID=42):
GET /api/users/42/orders
Authorization: Bearer <token_user42>
→ 200 OK: [{order_id:1, total:€50,...}]

# IDOR attack (Python script):
for uid in range(1, 10000):
    r = requests.get(f'/api/users/{uid}/orders',
        headers={'Authorization':'Bearer <token>'})
    if r.status_code == 200:
        save_data(uid, r.json())  # leak!

# No errors, no alerts in the logs
# 10,000 users exposed in minutes
```

✓ Mitigation: Always verify on the server that the authenticated user owns the specific resource (not just that they are authenticated)

OWASP Top 10 attacks (examples)

A02 – Cryptographic Failures

What is it?
Sensitive data is exposed due to missing, weak, or incorrectly used cryptography (both in transit or at rest)

How it works
Weak password hashing
Data sent over HTTP
Hardcoded secrets

Typical attacks
Password cracking
Man-in-the-Middle
Token theft

Impact
Credential compromise
Financial and privacy damage

✅ **Mitigation:** Use modern crypto (TLS, Argon2/bcrypt) and proper key management.

Data at risk may include passwords, payment data, medical records, API keys, session tokens, ..

Most common mistakes:

- Passwords stored with MD5/SHA-1s
- Data transmitted via HTTP (no TLS) — interceptable on public Wi-Fi
- Deprecated algorithms: DES, RC4, MD5, SHA-1 — considered broken
- Hardcoded encryption keys in source code — exposed in git repositories
- Insecure random number generation

OWASP Top 10 attacks (examples)

A02 – Cryptographic Failures

Example: Password Hash Cracking — MD5

The application stores passwords using MD5 without salt. After a data breach (e.g. via SQL Injection), the attacker obtains all hashes. Tools like hashcat with a modern GPU can reverse MD5 at 164 billion hashes/second, cracking common passwords in milliseconds.

Attack Steps:

1. Attacker obtains a database dump (e.g. via SQLi or insider breach)
2. Identifies the hash algorithm by format (MD5: 32 hex chars, SHA1: 40 chars)
3. Runs hashcat with a wordlist (rockyou.txt: 14M passwords) + mutation rules
4. Cracks 80%+ of passwords in under one hour
5. Uses credentials for credential stuffing on other services (password reuse)

Payload / Code

```
# MD5 hash of password '123456' (no salt):
MD5('123456') → e10adc3949ba59abbe56e057f20f883e

# Attack with hashcat (GPU RTX 4090):
hashcat -m 0 -a 0 hashes.txt rockyou.txt
--rules-file best64.rule

# Speed: 164,000 MH/s (164 billion/s)
# Result in < 1ms:
e10adc3949...  →  '123456'
5f4dcc3b5aa76...  →  'password'

# With bcrypt (cost=12): 100 hashes/s
# Protection increase: 1,640,000,000×
```

✓ Mitigation: Use Argon2id (NIST 2024 recommended) or bcrypt (cost≥12) for passwords. NEVER MD5/SHA-1. Enforce HTTPS.

OWASP Top 10 attacks (examples)

A03 – Injection

What is it?
Untrusted input is interpreted as commands or code by an interpreter

How it works
Input is concatenated into queries or commands
No validation or parameterization

Typical attacks
SQL Injection
XSS
Command Injection

Impact
Data theft
Remote Code Execution

✅ **Mitigation:** Use parameterized queries and encode output

OWASP Top 10 attacks (examples)

A03 - Injection

Example: SQL Injection — Authentication Bypass, Data Dump

The SQL query is built by directly concatenating user input. The attacker injects SQL metacharacters that alter the query logic

Attack Steps:

1. Attacker tries and identifies vulnerable parameters (', ", ;, OR 1=1, SLEEP())

2. Authentication bypass: uses 'admin'--' into the login field

3. Uses UNION SELECT to get list of users and passwords

Payload / Code

```
# Código vulnerável (python)
query = "SELECT * FROM users WHERE username = '" +
username + "'"

#Se o utilizador escrever admin, a query fica:
SELECT * FROM users WHERE username = 'admin'

# Mas se escrever admin' OR '1'='1:
SELECT * FROM users WHERE username = 'admin' OR '1'='1'
```

✓ Mitigation: using parameterized (prepared) queries, avoiding direct input concatenation, validating inputs, and enforcing least-privilege database access.

OWASP Top 10 attacks (examples)

A03 – Injection
XSS

What is it?
XSS occurs when a web application includes unsanitized user input in a page, allowing malicious JavaScript to run in the victim's browser.

How it works

Attacker injects script via input, URL, or stored data
Application reflects or stores the input without proper encoding
Browser executes the script as trusted site content

Main types

Reflected XSS – payload reflected in the HTTP response
Stored XSS – payload saved and executed for every user
DOM-based XSS – vulnerability only in client-side JavaScript

Impact

Session and cookie theft, phishing and redirects actions performed as the victim

OWASP Top 10 attacks (examples)

A03 - Injection

Reflected XSS

XSS occurs when an application includes user-supplied data in an HTML page without proper encoding, allowing scripts to execute in other users' browsers. In 2021, OWASP merged XSS into A03:Injection — it is the most prevalent client-side attack
In reflected XSS, the attacker sends a malicious link (often via email) that reflects injected JavaScript in the HTTP response, which is then executed in the victim's browser.

Attack Steps:

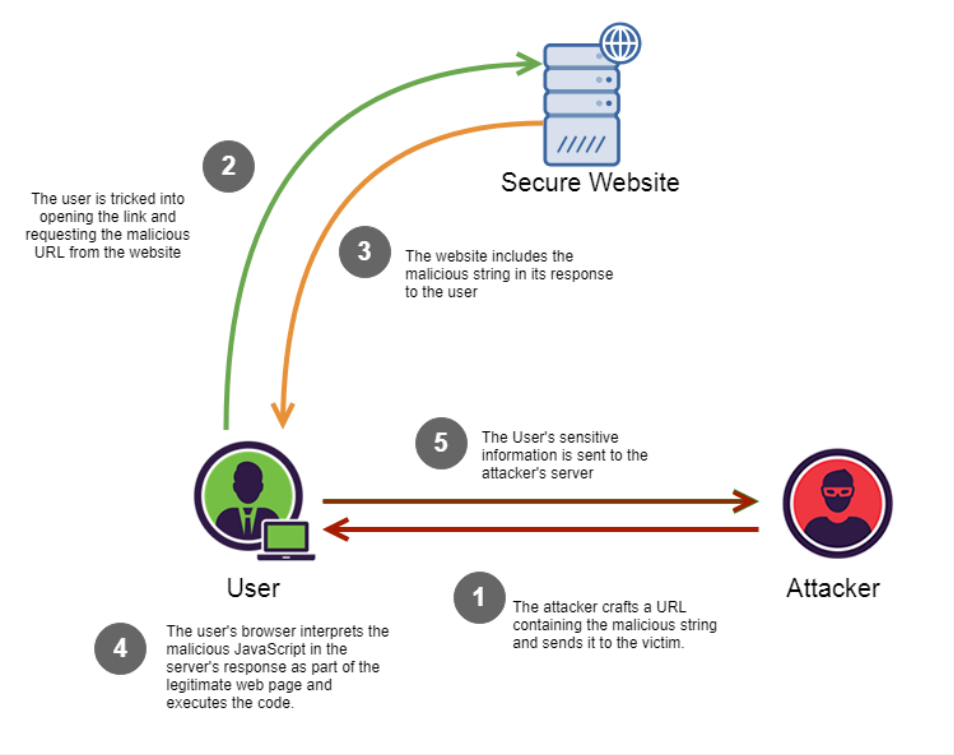
1. Attacker crafts a malicious URL containing injected JavaScript in a request parameter.

2. Victim clicks the link (often received via email, message, or social media).

3. Vulnerable server reflects the input in the HTTP response without proper encoding.

4. Victim's browser executes the script as trusted code from the website.

5. Attacker exploits the result, such as stealing session cookies or performing actions as the victim.



OWASP Top 10 attacks (examples)

A03 - Injection

Stored XSS

Stored XSS occurs when malicious user input is saved on the server (e.g., in a database) and automatically executed in the browsers of all users who view the affected page

Attack Steps:

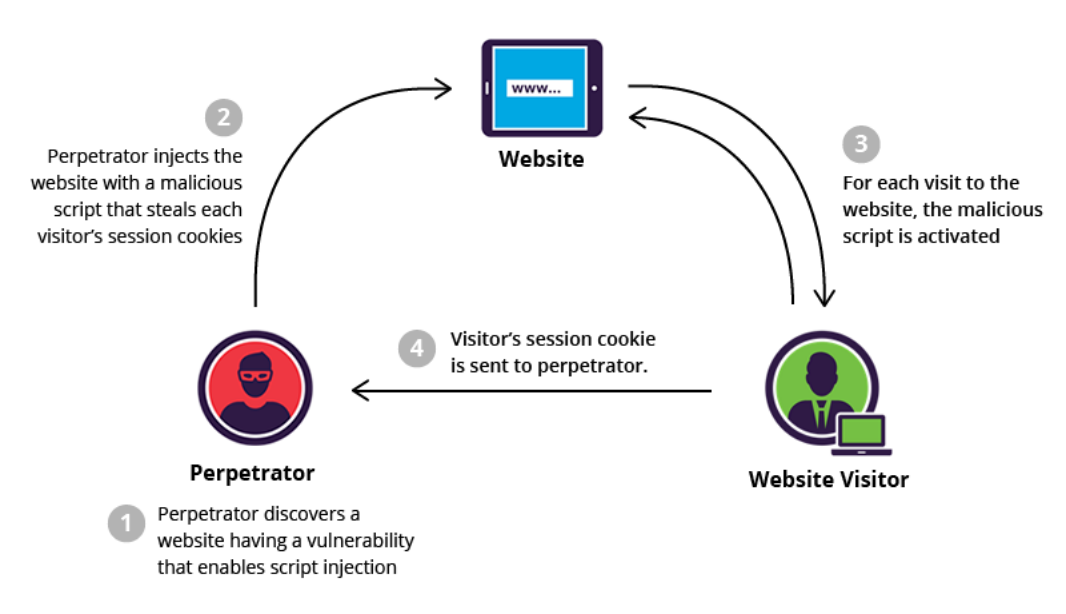
1. The attacker submits JavaScript in a persistent field (comments, profile, posts).

2. The application stores the input without proper validation or encoding.

3. The stored content is included in server responses to users.

4. Victims' browsers execute the script as trusted site code.

5. The attacker exploits the result (session theft, actions as victims, worms).



OWASP Top 10 attacks (examples)

A03 - Injection

DOM-Based XSS

DOM-based XSS occurs when client-side JavaScript takes untrusted data (for example from the URL or user input) and writes it into the DOM in an unsafe way, causing malicious JavaScript to execute in the victim's browser.
The server is not involved in injecting the payload — the vulnerability exists entirely in the browser.

Attack Steps:

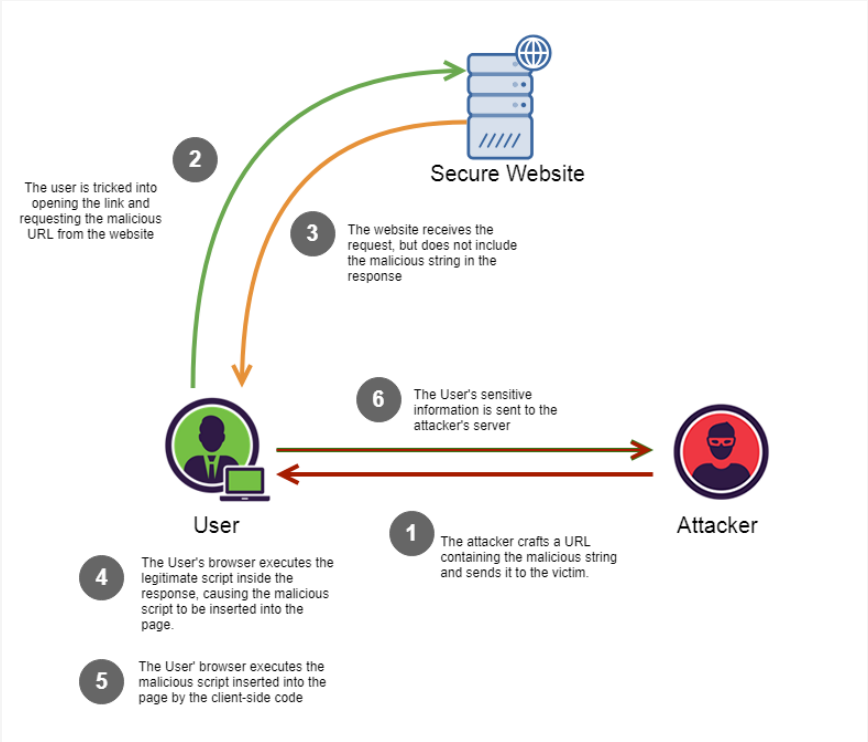
1. The attacker submits JavaScript in a persistent field (comments, profile, posts).

2. The application stores the input without proper validation or encoding.

3. The stored content is included in server responses to users.

4. Victims' browsers execute the script as trusted site code.

5. The attacker exploits the result (session theft, actions as victims, worms).



OWASP Top 10 attacks (examples)

A03 – Injection
XSS - Mitigation

✅ **Mitigation**

Context-aware output encoding
Encode user input when rendering it in HTML, JavaScript, or URLs so it is treated as data, not code.

Avoid unsafe DOM APIs
Do not use innerHTML, eval, document.write, or similar functions with user-controlled data; use safe alternatives like textContent.

Server-side input validation
Validate input type, length, and format to reject unexpected or malicious data early (defense in depth).

Content Security Policy (CSP)
Restrict which scripts can execute in the browser to reduce the impact of XSS, even if a vulnerability exists. Defines and sent by the server

Sanitisation for rich content
When HTML is allowed (e.g. comments), remove dangerous tags and attributes using allowlists.

Use secure frameworks and templates
Prefer frameworks that escape output by default and reduce manual handling errors.

OWASP Top 10 attacks (examples)

A04 – Insecure Design

Example: Missing Rate Limiting — PIN/OTP Brute Force and User Enumeration

Insecure Design refers to architectural flaws that cannot be fixed by code alone — they require redesign. The absence of rate limiting on an authentication flow enables brute force attacks. Non-uniform error responses reveal whether a user account exists (enumeration).

Attack Steps:

1. Application does not limit login attempts per IP or per account
2. Different responses for 'user not found' vs 'wrong password' — enumeration
3. Attacker enumerates valid users using a list of common emails
4. For each valid user, executes brute force with a password wordlist
5. A 4-digit PIN (10,000 combinations) discovered in < 10 seconds without rate limiting

Payload / Code

```
# User enumeration (different response):
POST /login {"user":"admin@x.com"}
→ 'Incorrect password' (user exists!)
POST /login {"user":"xyz@x.com"}
→ 'User not found'

# OTP brute force (6 digits):
for pin in range(000000, 999999):
    r = post('/verify', {'otp': pin})
    if 'success' in r.text: break
# No rate limit: 1000 req/s
# = 999999 attempts in ~17 min

# 4-digit PIN: cracked in ~10s!
```

✓ Mitigation: Rate limiting (e.g. 5 attempts, 15-min lockout). CAPTCHA after N failures. Real-time alerts for anomalous attempts.

OWASP Top 10 attacks (examples)

A05 – Security Misconfiguration

Example: Exposed Sensitive Files + phpMyAdmin with Default Credentials

Incorrect web server configuration exposes directory listing and sensitive files (.env, .git, DB dumps). The admin panel is publicly accessible with factory-default credentials. In production environments, default configurations are frequently forgotten. This type of flaw is trivially exploitable and automatically detected by scanners.

Attack Steps:

1. Scanner (Nikto/gobuster) finds /backup/, /.env, /.git/, /phpmyadmin/
2. Directory listing exposes database_dump_2024.sql and configuration files
3. The .env file contains DB credentials, AWS keys and API tokens in plaintext
4. phpMyAdmin publicly accessible with root:root (default credentials)
5. Attacker gains full DB control and cloud account access in minutes

Payload / Code

```
# Automated reconnaissance:
gobuster dir -u https://target.com
  -w /usr/share/wordlists/common.txt
→ /backup/ (200) → Index of /backup/
   database_dump_2024.sql [2.3 GB]
→ /.env (200)
→ /phpmyadmin/ (200)

# Contents of exposed .env:
DB_HOST=db.internal.com
DB_PASSWORD=SuperSecret123
AWS_ACCESS_KEY=AKIAIOSFODNN7EX
AWS_SECRET=wJalrXUtnFEMI/K7MDENG
STRIPE_KEY=sk_live_aBcDeFg123...

# phpMyAdmin → root:root → full access
```

✓ Mitigation: Disable directory listing (Options -Indexes). Block .env, .git, /backup in the web server. Change ALL default credentials before going to production. Run regular scanners (OWASP ZAP). Remove administration panels from production or restrict by IP/VPN.

OWASP Top 10 attacks (examples)

A07 – Identification & Authentication Failures

Example: JWT with Algorithm 'none' — Complete Authentication Bypass

JSON Web Tokens (JWTs) are widely used for stateless authentication. If the server accepts the 'none' algorithm (no signature), any user can forge a JWT with any claims they want, including admin claims, without knowing the secret key.

Attack Steps:

1. User authenticates and receives a valid JWT (e.g. alg=HS256)
2. Decodes the JWT in Base64 — readable without the secret key
3. Changes the header to alg=none and the payload to role=admin
4. Removes the signature (keeps the trailing dot) — forged JWT ready
5. Vulnerable server accepts the JWT without verifying signature → admin access

Payload / Code

```
# Legitimate JWT (3 parts: header.payload.sig):
eyJhbGciOiJIUzI1NiJ9.        ← header
eyJ1c2VyIjoiam9hbyIsInJvbGUi.← payload
Sig_HMAC_here                 ← signature

# Decoded payload:
{"user":"john", "role":"user"}

# Attack: change to alg=none:
eyJhbGciOiJub25lIn0.          ← alg=none
eyJ1c2VyIjoiYWRtaW4iLCJyb2xl.← role=admin
                              ← no signature

# Vulnerable server: 200 OK → admin!
```

✓ Mitigation: NEVER accept alg=none. Use a well-maintained JWT library and explicitly configure the expected algorithm (e.g. RS256). Implement short expiry (15 min) + refresh tokens with rotation. Validate ALL claims. MFA for privileged accounts. Invalidate tokens on logout.

OWASP Top 10 attacks (examples)

A08 – Software & Data integrity Failures

Example: Supply Chain Attack — Compromised npm Package

An attacker compromised the npm maintainer account of the 'event-stream' package and published a version with a hidden malicious dependency (flatmap-stream). The obfuscated and encrypted code only activated in environments with the Bitpay Bitcoin wallet (copay). It affected 8 million downloads before being detected.

Attack Steps:

1. Attacker offers to 'help' maintain a popular package (social engineering)
2. Maintainer transfers npm account control — attacker publishes malicious version
3. Version with backdoor distributed as a transitive dependency (invisible to the dev)
4. Obfuscated and encrypted code detects the target environment
5. Exfiltrates private Bitcoin keys to the attacker's server

Payload / Code

```
# package.json of the victim's project:
{
  "dependencies": {
    "event-stream": "4.0.0"
  }  // pulls in malicious flatmap-stream
}

# Malicious code (deobfuscated):
const r = require,  // obfuscated
s = r('crypto');
function decrypt(e, n) {
  // decrypts payload at runtime
  // only activates if Bitpay wallet found
  steal_private_keys(wallet);
}

# Detection: npm audit
→ Critical: event-stream@4.0.0
```

✓ Mitigation: Verify package integrity, review maintainer permissions for critical dependencies

OWASP Top 10 attacks (examples)

A09 – Security Logging & Monitoring Failures

Example: Silent Attack

Without adequate logging and monitoring, attacks go undetected. The industry average for breach detection is 207 days (IBM Cost of Data Breach 2023). Additionally, if logs accept unsanitised input, an attacker can inject false entries to cover their tracks or create a false alibi.

Attack Steps:

1. Attacker attempts 50,000 login attempts over 3 weeks (slow and low)
2. No alerts configured for login failure spikes — completely unnoticed
3. Compromises an account and exfiltrates data over weeks
4. Injects lines into the logs to erase tracks: \n[INFO] All systems normal
5. Organisation detects the breach 6 months later in a manual audit

Payload / Code

```
# Log Injection – erasing tracks:
# Malicious input in the username field:
admin\n2024-01-15 INFO [Auth] User
  'admin' logged in successfully
  from 10.0.0.1

# Resulting log (falsified):
[WARN] Failed login: 'admin\n'
[INFO] [Auth] User 'admin' logged in
  successfully from 10.0.0.1

# 'Low and slow' attack with no alerts:
50,000 attempts over 3 weeks
= ~240 per hour = 4 per minute
→ passes as 'normal traffic'
```

✓ Mitigation: Sanitise input before writing to logs (escape \n, \r, control characters). Configure alerts for N failed logins in X minutes, off-hours access, data exfiltration. Centralise logs in a SIEM (ELK, Splunk). MTTD < 1h as a KPI. Immutable logs with a minimum 12-month retention.

OWASP Top 10 attacks (examples)

A10 – Server-Side Request Forgery (SSRF)

Example: SSRF to AWS Metadata Service — IAM Credential Theft

A legitimate 'URL preview' feature causes the server to fetch external content. The attacker redirects this functionality to internal IPs that are not publicly accessible. In cloud environments (AWS), the 169.254.169.254 endpoint exposes temporary IAM credentials that give full control of the cloud account. The Capital One breach (2019) was caused by SSRF.

Attack Steps:

1. Application offers a preview/fetch feature for external URLs
2. Attacker submits an internal URL: http://169.254.169.254/latest/meta-data/
3. Server makes the request internally — response returned to the attacker
4. Navigates to /iam/security-credentials/<role-name> → obtains AccessKey + Secret
5. Uses credentials to access S3, RDS, EC2 — full control of the cloud infrastructure

Payload / Code

```
# Legitimate feature:
POST /api/preview
{"url":"https://example.com"}
→ 200: {"title":"Example Domain",...}

# SSRF attack:
POST /api/preview
{"url":"http://169.254.169.254/latest/
  meta-data/iam/security-credentials/
  EC2-Production-Role"}

# Server response:
{"AccessKeyId":"ASIA4K6EXAM...",
 "SecretAccessKey":"wJalrXUtnFEM...",
 "Token":"AQoD...HkM9IEXAMPLE...",
 "Expiration":"2024-01-16T12:00:00Z"}
# Attacker has full cloud access!
```

✓ Mitigation: Validate all user-supplied URLs and use an allowlist of permitted domains/IPs. Block RFC 1918 and 169.254.0.0/16 at the network layer.

WSTG – Web security testing guide

- Help in the approach to systematically test a web application
- Defines test categories

What is it?

The OWASP reference guide for security testing. Published in 2021 (v4.2), with 91 test cases organised into 12 categories.

Each test case includes:
- Unique ID (e.g. WSTG-INPV-01)
- Test objectives
- How to test (techniques)
- Recommended tools
- Remediation

The ID is structured as:
WSTG-[CATEGORY]-[NUMBER]

12 Test Categories

INFO	Information Gathering	INPV	Input Validation
CONF	Configuration & Deployment	ERRH	Error Handling
IDNT	Identity Management	CRYP	Cryptography
AUTH	Authentication Testing	BUSL	Business Logic
ATHZ	Authorization Testing	CLNT	Client-Side Testing
SESS	Session Management	APIT	API Testing

Security identifiers: Purpose & Scope

WSTG (OWASP): Web Security Testing Guide

- Identifies security test cases
- Format: WSTG-CATEGORY-NN
- Example: WSTG-INPV-01 — Input Validation test
- ✅ How to test (testing methodology)

OWASP Top 10

- Identifies web application risk categories
- Format: A01: Broken Access Control
- ✅ Risk communication and prioritisation

CVE: Common Vulnerabilities and Exposures

- Identifies a real, specific vulnerability
- Format: CVE-YYYY-NNNNN
- ✅ Public disclosure, patching, scanners

CWE

- Common Weakness Enumeration
- Identifies types of software weaknesses
- Examples: CWE-79 (XSS), CWE-89 (SQL Injection)
- ✅ Root cause at code or design level

CAPEC

- Common Attack Pattern Enumeration and Classification
- Identifies attack patterns
- Example: CAPEC-66 — SQL Injection
- ✅ Describes how attackers exploit weaknesses

CVSS

- Common Vulnerability Scoring System
- Numerical severity score (e.g. 8.8 – High)
- ✅ Used to prioritise remediation

How identifiers fit together

- A single security issue can have multiple identifiers, each serving a different goal: testing, classification, disclosure, or risk assessment.
- CWE → CAPEC → CVE → OWASP Top 10 / CVSS

OWASP ZAP (tool-level identifiers)

When ZAP reports an issue, it typically includes:

- ZAP Alert ID — internal identifier (e.g. 40012)
- OWASP Top 10 mapping
- CWE (primary weakness)
- WASC (legacy category, if applicable)
- CVSS score (when available)
- ✅ ZAP correlates multiple standards for the same issue

Summary

- What is a web application
- Why are applications target of attacks?
- Attack surface of a web application
- HTTP itself is the attack vector
- OWASP project
- OWASP Top 10 risk categories
- OWASP Top 10 attacks
- WSTG
- Security Identifiers

Useful Links

OWASP: https://owasp.org/

OWASP ZAP: https://www.zaproxy.org/

OWASP WSTG: https://github.com/OWASP/wstg/tree/master/document

OWASP Top 10: https://owasp.org/Top10/2025/

CVE: Common Vulnerabilities and Exposures: https://www.cve.org/

CWE: Common Weknesses Enumeration: https://cwe.mitre.org/

CAPEC: https://capec.mitre.org/

CVSS: https://www.cve.org/about/relatedefforts