

# Web and Mobile Application Testing

---

Martin Stanek

2024

# Table of Contents

Web application testing

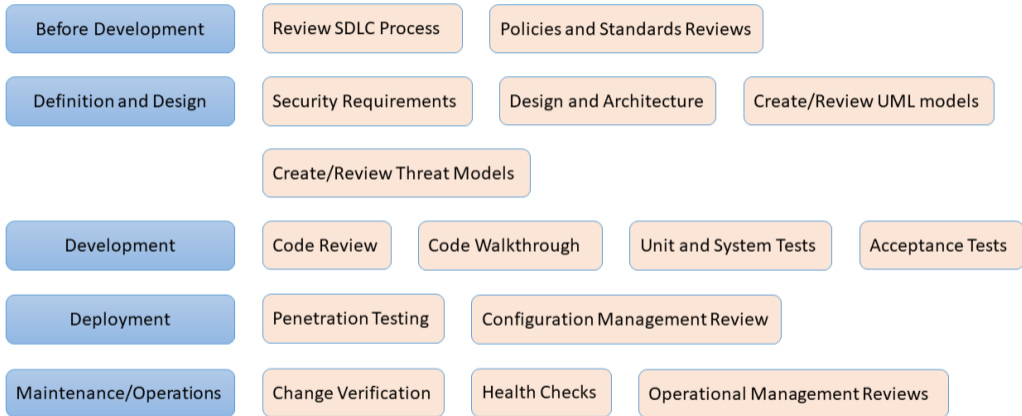
Mobile application testing

# Web applications and Mobile applications

- very frequent target (scope) of security testing activities
  - penetration testing, code review, scanning etc.
- unique business requirements
  - custom design and development
  - custom code, multiple components, libraries  $\Rightarrow$  opportunity for vulnerabilities

- OWASP Web Security Testing Guide
  - current stable version: 4.2 (2020)
  - **the guide** for penetration testing of web applications
- much more than guidance on web app penetration testing; included:
  - comprehensive introduction
  - OWASP testing framework (phases, SDLC testing workflow)
  - security and reviewing/testing through the SDLC
- WSTG describes
  - why to test (summary)
  - how to test
  - tools when applicable
  - remediation
- this lecture: selected examples from each section

# SDLC Workflow



- OWASP Application Security Verification Standard (version 4.0.3, 2021)
- security requirements (“Verify that . . .”) in 14 areas (controls)
  - V1 Architecture, Design and Threat Modeling . . . V14 Configuration
- three security verification levels:
  - Level 1: low assurance levels, completely penetration testable
  - Level 2: applications that contain sensitive data (recommended)
  - Level 3: the most critical applications
- majority of requirements in levels 2 and 3 require access to
  - documentation, source code, configuration, and the people involved in the development process
- use for security testing
  - specification of what should be tested (what)
  - does not tell you how to test

# WSTG Structure

1. Information Gathering
2. Configuration and Deployment Management Testing
3. Identity Management Testing
4. Authentication Testing
5. Authorization Testing
6. Session Management Testing
7. Input Validation Testing
8. Testing for Error Handling
9. Testing for Weak Cryptography
10. Business Logic Testing
11. Client-side Testing
12. API Testing

# Information Gathering

- fingerprinting (web app, web server, framework)
- metadata, application entry points, enumerate applications
- web page information leaks

(WSTG-INFO-01) Conduct Search Engine Discovery Reconnaissance for Information Leakage

objective: identify sensitive information exposed directly or indirectly

- check `robots.txt`
- use search engines (Google, Shodan, etc.)
- employ various search operators (Google Hacking Database, etc.)
- view archived content



## Information Gathering (2)

(WSTG-INFO-08) Fingerprinting Web Application Framework

objective: identify the framework (WordPress, Django, ZK, etc.)

- HTTP headers (X-Powered-By in the HTTP response)
- Cookies (fe\_typo\_user for TYPO3)
- HTML source code (framework and version in comments for generated HTML, framework-specific paths for CSS and JS)
- Specific files and folders (/wp-includes/, /wp-admin/ with HTTP response codes indicating their existence)
- File extensions (.php, .aspx)
- Error messages (if not sanitized)

# Configuration and Deployment Management Testing

- network infrastructure, application platform
- sensitive information – file extensions handling, old backups, unreferenced files
- admin interfaces, HTTP methods, HSTS, RIA cross domain policy, file permissions, subdomain takeover, cloud storage

## (WSTG-CONF-08) Test HTTP Strict Transport Security

- HSTS instructs client to never connect through unencrypted HTTP
- test HSTS header (`Strict-Transport-Security`)

## Configuration and Deployment Management Testing (2)

### (WSTG-CONF-09) Test File Permission

objective: review and identify any rogue file permissions

- where to look (files and directories):
  - web files, configuration, sensitive files (encrypted data, password, key)
  - database files, log files (security, operation, admin),
  - executables (scripts, EXE, JAR, class, PHP, ASP)
  - temp, upload
- tools mentioned: AccessEnum, AccessChk (Windows), namei (Linux)

# Identity Management Testing

- role definitions, user registration, account provisioning
- account enumeration and guessable accounts
- weak or unenforced username policy

## (WSTG-IDNT-02) Test User Registration Process

objective: verify that the process is aligned with security requirements

- Can anyone register? Is there a vetting by a human?
- Can a single user register multiple times?
- What identity proof is required?

## Identity Management Testing (2)

(WSTG-IDNT-04) Account Enumeration and Guessable User Account

objective: review user identification process and enumerate users through response analysis

- compare server response to valid username and password with
  - valid username & wrong password
  - nonexistent username
- different error codes, messages, or timing can reveal sensitive information
- guessing usernames (for usernames with an internal structure)

# Authentication Testing

- encrypted channel, default credentials, weak lockout mechanism
- weak password policy, security questions, password change/reset mechanism
- browser cache, weak authentication in alternative channel

## (WSTG-ATHN-02) Testing for Default Credentials

objective: enumerate default credentials and validate if they still exist

- test default accounts based on apps and frameworks identification results
- test obvious account names and passwords (admin, system, operator etc.)
- test for default passwords for new user accounts

## Authentication Testing (2)

### (WSTG-ATHN-04) Bypassing Authentication Schema

objective: verify the authentication is used wherever needed

- forced browsing (direct URL input), parameter modification (somepage?authenticated=1)
- predictable session ID, SQL injection in authentication form
- grey-box testing: review the code for authentication

# Authorization Testing

- file inclusion directory traversal, bypassing authorization schema
- privilege escalation, IDOR (Insecure Direct Object References)

## (WSTG-ATHZ-01) Directory Traversal File Include

objective: identify input vectors, assess bypassing techniques

- identify: filename in a request (interesting variable name, extension), in cookie
- testing various payloads:

```
foo.bar/getUserProfile.jsp?item=../../../../etc/passwd
```

```
Cookie: USER=1826cc8f:PSTYLE=../../../../etc/passwd
```

```
foo.bar/index.php?file=http://192.168.0.2:9080
```

- URL, UTF-8 and other encodings to bypass filters (e.g., %2e%2e%2f represents ../)



## Authorization Testing (2)

(WSTG-ATHZ-04) Insecure Direct Object References

objective: identify points where IDOR may occur, test for vulnerability

- direct use of a parameter
  - to access DB record, file resource, e.g.  
`foo.bar/somepage?invoice=12345`  
`foo.bar/somepage?img=img12345`
  - to perform operation in the system, e.g.  
`foo.bar/changepassword?user=someuser`
  - to access functionality  
`foo.bar/performpage?task=delete`

# Session Management Testing

- session management, cookies attributes, session fixation
- exposed session variables, CSRF
- logout functionality, session timeout/puzzling/hijacking

## (WSTG-SESS-02) Cookies Attributes

objective: proper security configuration for cookies

- attributes: Secure, HttpOnly, Path, SameSite, Domain, Expires
- specific semantic for name prefixes: `__Secure-`, `__Host-`
  - imply required attributes

## (WSTG-SESS-05) Cross Site Request Forgery (CSRF)

- request to web application (site) where user is already authenticated
  - session cookie/credentials are used and server trusts this (authorized) request
  - requests that change state (e.g. delete, modify, create)



- prevention typically includes CSRF tokens (per-request, per-session)
  - included for state-changing pages/operations (e.g. in a hidden HTML form parameter)

# Input Validation Testing

- XSS (reflected and stored)
- injection (SQL, LDAP, XML, SSI, XPath, IMAP/SMTP, code, local/remote file, host header, server-side template)
- HTTP (splitting, smuggling, parameter pollution)

## (WSTG-INPV-01) Reflected Cross Site Scripting

objective: identify reflected variables in the responses, asses encoding applied (of any)

- executing malicious code inside a victim's browser
  - manipulate what user sees and interacts with, retrieve and modify data
- reflected: data from a request immediately unsafely reflected in the response  
`foo.bar/index.php?user=<script>alert(123)</script>`
- bypassing ad-hoc encoding

## (WSTG-INPV-05) SQL Injection

objective: identify injection points, and asses the impact (obligatory [xkcd reference](#))

```
SELECT * FROM users WHERE username='$username' AND password='$password'
```

parameters: username = admin'-- and empty (arbitrary) password

- detection: disrupt the query ' , " , ` , ' ) , " ) , etc.
- database dependent (syntax, functions)
  - database identification – different functions available
- different types of SQL injections
  - in-line (you get the response): error-based, union-based `SELECT Name, Phone, Address FROM Users WHERE Id=$id $id=1 UNION ALL SELECT creditCardNumber,1,1 FROM CreditCardTable`
  - blind (you can infer an information): boolean-based, time-based
    - out of band – initiate a separate connection (using DB functions)
- automated exploitation: sqlmap

# Testing for Error Handling

- improper error handling

## (WSTG-ERRH-01) Improper Error Handling

objective: trigger and analyze errors

- identify data entries and expected data types and formats
  - wrong inputs (manual approach), fuzzing (if possible), refine accordingly
- sources: stack traces, error responses (pages), HTTP responses, etc.
- API structure, version information for components, paths, etc.
- mishandled errors are DoS possibility

# Testing for Weak Cryptography

- weak TLS, padding oracle, weak encryption
- sensitive information sent via unencrypted channels

## (WSTG-CRYP-01) Weak Transport Layer Security

objective: verify the configuration and certificates

- TLS configuration (versions, ciphers)
- certificates (valid and trusted)
- redirect HTTP to HTTPS, HSTS with preload
- tools: SSL Labs, testssl.sh (and many others)

## Testing for Weak Cryptography (2)

### (WSTG-CRYP-03) Sensitive Information Sent via Unencrypted Channels

objective: identify sensitive information and assess the security and privacy of the channels used

- form data and cookies transmitted through HTTP
  - credentials, session ID, credit card numbers, personal information, etc.
- source code, configuration files, log files
  - passwords, encryption keys, API keys etc.



# Business Logic Testing

- business logic data validation, forging requests, integrity checks
- process timing, limits for number of times a function can be used
- circumvention of work flows, application misuse
- uploads: unexpected file types, malicious files (shells, malware)

## (WSTG-BUSL-02) Ability to Forge Requests

- objectives:
  - review guessable, predictable, or hidden functionality of fields
  - insert logically valid data in order to bypass normal business logic workflow
- intercept POST/GET HTTP requests and change the parameters
  - skipping or bypassing steps in a workflow
  - hidden or undocumented features, for example debugging

### (WSTG-BUSL-09) Upload of Malicious Files

objective: assess ability to upload malicious file and get it processed

- malicious files: web shells (command execution by the server), malware
- bypassing poorly implemented file extension filter
- EICAR test file:  
`X50!P%@AP[4\PZX54(P^)7CC)7}$EICAR-STANDARD-ANTIVIRUS-TEST-FILE!$H+H*`
- xml (XXE - XML eXternal Entities), office documents, etc.
- DoS:
  - zip bomb (short archive decompressing into a huge output file)
  - billion laughs attack (parsing XML file)

# Client-side Testing

- DOM-based cross site scripting, JavaScript execution, HTML injection
- client-side URL redirect, CSS injection, resource manipulation
- Cross Origin Resource Sharing, cross site flashing, cross site script inclusion
- Clickjacking, WebSockets, web messaging, browser storage

## (WSTG-CLNT-04) Client-side URL Redirect

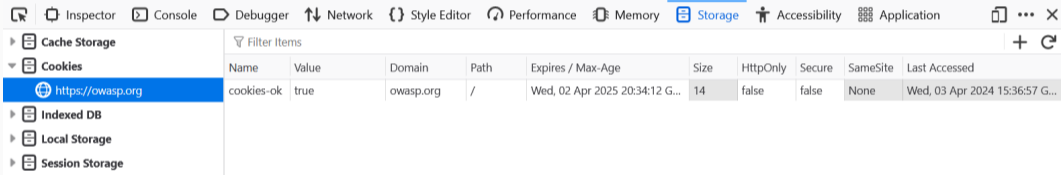
objective: identify where URL can be injected, assess locations where client could redirect to

- redirect to malicious site (e.g., using JavaScript `window.location`)
- potentially abused by phishing (originating on trusted site), stealing credentials

# Client-side Testing (2)

## (WSTG-CLNT-12) Browser Storage

objective: verify if sensitive data is stored in client-side storage



The screenshot shows the Chrome DevTools Storage Inspector interface. The 'Storage' tab is active, and the 'Cookies' section is expanded for the domain 'https://owasp.org'. A table displays the following cookie data:

Name	Value	Domain	Path	Expires / Max-Age	Size	HttpOnly	Secure	SameSite	Last Accessed
cookies-ok	true	owasp.org	/	Wed, 02 Apr 2025 20:34:12 G...	14	false	false	None	Wed, 03 Apr 2024 15:36:57 G...

- client-side storage: Local Storage, Session Storage, IndexedDB, Cookies
- *look how google.com uses client-side storage*

- GraphQL

(WSTG-APIT-01) GraphQL

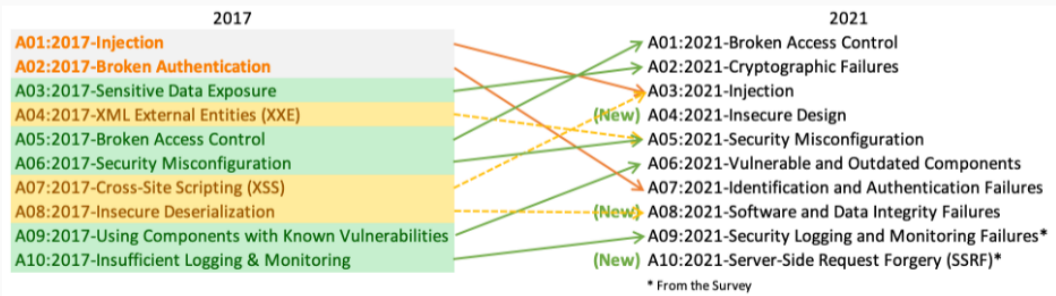
objectives:

- verify deployed configuration
- test all input fields for generic attack (proper input validation)
- validate proper access control
  - for example to introspection queries (schema, what queries are supported etc.)

- penetration test without report is (almost) useless
- areas that should be covered:
  - scope, methodology, timeline
  - executive summary (found vulnerabilities, severity, brief description of impact)
  - details of findings – show the vulnerability, suggest possible remediation
  - severity – combination of likelihood (exploitability) and impact
- review
  - comments, opposing views finding/remediation, additional information

# OWASP Top 10 (2021)

- primarily an **awareness** document
- top 10 risks – not a checklist for penetration test
- not an application security standard – use ASVS if you need one



## OWASP Top 10 (2021) – cont.

- 8 categories based on contributed data and analysis (highest incidence rates)
  - testing vendors, bug bounty vendors, organizations with internal testing data
  - more than 500,000 applications
  - incidence rate  $\sim$  the percentage of applications vulnerable to that CWE from the population tested by that org for that year.
- 2 categories (not already present) from a community survey
- each category is defined by a list of CWEs
  - although assigning vulnerability to CWE is not straightforward
- exploit and impact for each category based on CVE data from NVD
- OWASP plans to release Top 10:2024 in September 2024



- Burp Suite
  - limited community version
- OWASP ZAP (Zed Attack Proxy)
  - an alternative to Burp Suite
- standard features:
  - proxy (intercept and modify requests and responses)
  - automatic scans, passive scans
  - WebSocket support
  - brute-forcing, fuzzing
  - authentication and session management
- other tools: nikto, Arachni, . . .

## OWASP Top 10 API Security Risks – 2023

1. API1:2023 – Broken Object Level Authorization
2. API2:2023 – Broken Authentication
3. API3:2023 – Broken Object Property Level Authorization
4. API4:2023 – Unrestricted Resource Consumption
5. API5:2023 – Broken Function Level Authorization
6. API6:2023 – Unrestricted Access to Sensitive Business Flows
7. API7:2023 – Server Side Request Forgery
8. API8:2023 – Security Misconfiguration
9. API9:2023 – Improper Inventory Management
10. API10:2023 – Unsafe Consumption of APIs

# Vulnerable web applications

- [OWASP Vulnerable Web Applications Directory](#)
  - web and mobile applications registry
  - categories: Online, Offline, Mobile, Containerized (or VM)
- learn and practice
- vulnerable web applications:
  - WebGoat, Juice Shop, Damn Vulnerable Web Application (DVWA), etc.
  - crAPI (API focused vulnerable system)
- other vulnerable systems:
  - Metasploitable 3 (VM with many vulnerabilities)
  - Game of Active Directory (GOAD, vulnerable AD environment)

# Mobile application testing

- OWASP Mobile Application Security = MASVS + MASTG
- OWASP MASVS – Mobile Application Security Verification Standard
  - security standard for mobile apps
- OWASP MASTG – Mobile Application Security Testing Guide
  - testing guide
  - technical processes for verifying the controls listed in the OWASP MASVS
- MAS checklist, MAS testing profiles (L1, L2, R)

# MASTG Structure – General Testing Guide

1. Mobile Application Taxonomy
2. Mobile Application Security Testing
3. Mobile App Tampering and Reverse Engineering
4. Mobile App Authentication Architectures
5. Mobile App Network Communication
6. Mobile App Cryptography
7. Mobile App Code Quality
8. Mobile App User Privacy Protection

## MASTG Structure – Specific Testing Guides (Android/iOS)

1. Android/iOS Platform Overview
2. Android/iOS Security Testing
3. Android/iOS Data Storage
4. Android/iOS Cryptographic APIs
5. Android/iOS Local Authentication
6. Android/iOS Network Communication
7. Android/iOS Platform APIs
8. Android/iOS Code Quality and Build Settings
9. Android/iOS Anti-Reversing Defenses

Alternatives (choose one, extra credit for doing both):

1. TryHackMe: OWASP Top 10 – 2021
  - screenshot final stage for 3 different challenges proving their completion
2. Introduce 3 different vulnerabilities to a web application
  - use arbitrary open-source application
  - try to make vulnerabilities that could really happen (by omission, negligence or incompetence)
  - aim for different types of vulnerabilities and minimal changes in the source code
  - show how the vulnerabilities can be exploited

1. [OWASP Web Security Testing Guide, v4.2, 2020](#)
2. [OWASP Mobile Application Security Testing Guide, v1.7.0, 2023](#)