# OWASP Top Ten Cheat Sheet

From OWASP


# Introduction



The following is a developer-centric defensive cheat sheet for the 2013 release of the OWASP Top Ten Project. It also presents a quick reference based on OWASP Testing Project to help how to identify the risks.

# OWASP Top Ten Cheat Sheet

| | Presentation | Controller | Model | Testing (OWASP Testi |
|---|---|---|---|---|
| **A1 Injection** | *Render:*<br>▪ Set a correct content type<br>▪ Set safe character set (UTF-8)<br>▪ Set correct locale<br>*On Submit:*<br>▪ Enforce input field type and lengths.<br>▪ Validate fields and provide feedback.<br>▪ Ensure option selects and radio contain only sent values. | **Canonicalize using correct character set**<br>**Positive input validation using correct character set**<br>(NR) Negative input validation. (LR) Sanitize input.<br>*Tip: updating a negative list (such as looking for "script", "sCrIpT", "ßCrîpt", etc) will require expensive and constant deployments and will **always** fail as attackers work out your list of "bad" words. Positive validation is simpler, faster and usually more secure and needs updating far less than any other validation mechanism.* | ***Parameterized queries (https://www.owasp.org/index.php/Query_Parameterization_Cheat_Sheet)**<br>▪ Object relational model (Hibernate).<br>▪ Active Record design pattern.<br>▪ Stored procedures.<br>▪ Escape mechanisms such as ESAPI's Encoder:<br>  ▪ EncodeForLDAP()<br>  ▪ Encoder.EncodeforOS()<br>*Tip: **All** SQL Injection is due to dynamic SQL queries. Strongly consider prohibiting dynamic SQL queries within your organization* | *SQL Injection (OTG-IN*<br>*005%29) LDAP Injectio*<br>*INPVAL-006)) ORM Inj*<br>*(https://www.owasp.org*<br>*INPVAL-008) (https://w*<br>*INPVAL-009) (https://w*<br>*INPVAL-010) (https://w*<br>*Injection (OTG-INPVAL*<br>*011)) Code Injection (O*<br>*012)) Command Injectio*<br>*(https://www.owasp.org*<br>*INPVAL-014) (https://w* |
| | | *Design:*<br>▪ Only use inbuilt session management. | | *Test Role Definitions (O*<br>*Test User Registration I*<br>*(https://www.owasp.org*<br>*Process (OTG-IDENT-(*<br>*Testing for Account Enu*<br>*(https://www.owasp.org*<br>*004)) Testing for Weak (*<br>*(https://www.owasp.org*<br>*Credentials Transported*<br>*(https://www.owasp.org*<br>*001)) Testing for defaul*<br>*(https://www.owasp.org*<br>*mechanism (OTG-AUTH* |

| | | | | |
|---|---|---|---|---|
| **A2 Weak authentication and session management** | *Render:*<br>▪ Validate user is authenticated.<br>▪ Validate role is sufficient for this view.<br>▪ Set "secure" and "HttpOnly" flags for session cookies.<br>▪ Send CSRF token with forms. | ▪ Store secondary SSO / framework / custom session identifiers in native session object – do not send as additional headers or cookies.<br>▪ Validate user is authenticated.<br>▪ Validate role is sufficient to perform this action.<br>▪ Validate CSRF token. | Validate role is sufficient to create, read, update, or delete data<br>*Tip: Consider the use of a "governor" to regulate the maximum number of requests per second / minute / hour that this user may perform. For example, a typical banking user should not perform more than ten transactions a minute, and one hundred per second is dangerous and should be blocked.* | *AUTHN-003)) Testing fo (https://www.owasp.org password functionality ( (https://www.owasp.org Browser cache weaknes (https://www.owasp.org password policy (OTG-A AUTHN-007)) Testing fo (https://www.owasp.org password change or res (https://www.owasp.org Testing for Weaker auth (https://www.owasp.org Testing for bypassing au (https://www.owasp.org Privilege Escalation (O AUTHZ-003)) Testing fo (https://www.owasp.org attributes (OTG-SESS-0 for Session Fixation (O Testing for Exposed Ses (https://www.owasp.org Request Forgery (CSRF Testing for logout functi (https://www.owasp.org 007) (https://www.owasp SESS-008) (https://www* |
| **A3 XSS** | *Render:*<br>▪ Set correct content type<br>▪ Set safe character set (UTF-8)<br>▪ Set correct locale<br>▪ Output encode all user data as per output context<br>▪ Set input constraints<br>*On Submit:*<br>▪ Enforce input field type and lengths.<br>▪ Validate fields and provide feedback.<br>▪ Ensure option selects and radio contain only sent values. | **Canonicalize using correct character set**<br>**Positive input validation using correct character set**<br>(NR) Negative input validation (LR) Sanitize input<br>*Tip: Only process data that is 100% trustworthy. Everything else is hostile and should be rejected.* | *Tip: Do not store data HTML encoded in the database. This prevents new uses for the data, such as web services, RSS feeds, FTP batches, data warehousing, cloud computing, and so on.*<br>*Tip: Use OWASP Scrubbr to clean tainted or hostile data from legacy data* | *Testing for Reflected Cr (https://www.owasp.org Cross Site Scripting (OT (https://www.owasp.org Cross Site Scripting (OT based_Cross_site_scripti (https://www.owasp.org (OTG-CLIENT-003) (ht Cross Site Flashing (OT CLIENT-008))* |
| **A4 Insecure Direct Object References** | If data is from internal trusted sources, no data is sent.<br>Or<br>*Render:*<br>▪ Send indirect random access reference map value. | Obtain data from internal, trusted sources.<br>Or<br>Obtain direct value from random access reference access map. | Validate role is sufficient to create, read, update, or delete data. | *Testing Directory traver (https://www.owasp.org Direct Object Reference (https://www.owasp.org Local File Inclusion (ht Inclusion (https://www.o* |
| | Ensure web servers and application servers | Ensure web servers and application servers are hardened | | *Fingerprint Web Server Fingerprint Web Applic (https://www.owasp.org Application (OTG-INFO Network/Infrastructure ( (https://www.owasp.org* |

| | | | | |
|---|---|---|---|---|
| **A5 Security Misconfiguration** | are hardened.<br><br>PHP: Ensure allow_url_fopen and allow_url_include are both disabled in php.ini. Consider the use of Suhosin extension | XML: Ensure common web attacks (remote XSLT transforms, hostile XPath queries, recursive DTDs, and so on) are protected by your XML stack. Do not hand craft XML documents or queries – use the XML layer. | Ensure database servers are hardened | *Platform Configuration (https://www.owasp.org Handling for Sensitive I (https://www.owasp.org Review Old, Backup and (https://www.owasp.org CONFIG-004)) Enumer (https://www.owasp.org Test HTTP Methods (O Test RIA cross domain p (https://www.owasp.org ERR-001) (https://www. ERR-002) (https://www.* |
| **A6 Sensitive Data Exposure** | *Design:*<br><br>▪ Use strong ciphers (AES 128 or better) with secure mode of operations (do not use ECB).<br>▪ Use strong hashes (SHA 256 or better) with salts for passwords.<br>▪ Protect keys more than any other asset.<br>▪ Use TLS 1.2 or later for all web communications.<br>▪ Buy extended validation (EV) certificates for public web servers.<br><br>*Tip: Use TLS 1.2 always – even internally. Most snooping is done within corporate networks – and it is as easy and unethical as fishing with dynamite.*<br><br>*Render:*<br><br>▪ Do not send keys or hashes to the browser. | *Design:*<br><br>▪ Use strong ciphers (AES 128 or better) with secure mode of operations (do not use ECB).<br>▪ Use strong hashes (SHA 256 or better) with salts for passwords.<br>▪ Protect keys more than any other asset.<br>▪ Mandate strong encrypted communications between web and database servers and any other servers or administrative users.<br><br>*Tip: Only certain personally identifiable information and sensitive values MUST be encrypted. Encrypt data that would be embarrassing or costly if it was leaked or stolen.*<br><br>*Tip: It is best to encrypt data on the application server, rather than the database server.* | *Design:*<br><br>▪ Mandate strong encrypted communications with application servers and any other servers or administrative users.<br><br>*Tip: Do not use RDBMS database, row or table level encryption. The data can be retrieved in the clear by anyone with direct access to the server, or over the network using the application credentials. It might even traverse the network in the clear despite being "encrypted" on disk.* | *Testing for Weak SSL/T (https://www.owasp.org CRYPST-001)) Testing f (https://www.owasp.org sent via unencrypted cha (https://www.owasp.org 003)) Test HTTP Strict T (https://www.owasp.org Transported over an En (https://www.owasp.org 001))* |
| **A7 Missing Function Level Access Control** | *Design:*<br><br>▪ Ensure all non-web data is outside the web root (logs, configuration, etc).<br>▪ Use octet byte streaming instead of providing access to real files such as PDFs or CSVs or similar.<br>▪ Ensure every page requires a role, even if it is "guest".<br><br>*Pre-render:* | ▪ Validate user is authenticated.<br>▪ Validate role is sufficient to perform secured action.<br>▪ Validate CSRF token.<br><br>*Tip: It's impossible to control access to secured resources that the web application server does not directly serve. Therefore, PDF reports or similar should be served by* | Validate role is sufficient to create, read, update, or delete data | *Testing Directory traver (https://www.owasp.org authorization schema (C (https://www.owasp.org bypassing authenticatio (https://www.owasp.org* |

| | | | | |
|---|---|---|---|---|
| | ▪ Validate user is authenticated.<br>▪ Validate role is sufficient to view secured URL.<br><br>*Render:*<br><br>▪ Send CSRF token. | *the web application server using binary octet streaming.*<br><br>*Tip: Assume attackers **will** learn where "hidden" directories and "random" filenames are, so do not store these files in the web root, even if they are not directly linked.* | | |
| **A8 Cross Site Request Forgery** | *Pre-render:*<br><br>▪ Validate user is authenticated<br>▪ Validate role is sufficient for this view<br><br>*Render:*<br><br>▪ Send CSRF token.<br>▪ Set "secure" and "HttpOnly" flags for session cookies. | ▪ Validate CSRF token.<br>▪ Validate role is sufficient to perform this action.<br>▪ Validate role is sufficient.<br><br>*Tip: CSRF is **always** possible if there is XSS, so make sure XSS is eliminated within your application.* | Validate role is sufficient to create, read, update, or delete data | *Testing for Cross Site R (https://www.owasp.org* |
| **A9 Using Components with Known Vulnerabilities** | | | | *Enumerate Applications (https://www.owasp.org* |
| **A10 Unvalidated Redirects and Forwards** | ▪ Design the app without URL redirection parameters.<br><br>or<br><br>*Render:*<br><br>▪ Use random indirect object references for redirection parameters. | ▪ Design the app without URL redirection parameters.<br><br>or<br><br>▪ Obtain direct redirection parameter from random indirect reference access map.<br>▪ (LR) Positive validation of redirection parameter.<br>▪ (NR) Java – Do not forward() requests as this prevents SSO access control mechanisms. | ▪ Validate role is sufficient to create, read, update, or delete data. | *Testing for Client Side U (https://www.owasp.org* |

# Authors and Primary Editors

Andrew van der Stock vanderaj[at]owasp.org

Ismael Rocha Gonçalves ismaelrg[at]gmail.com

Jorge Correa jacorream@gmail.com

# Other Cheatsheets

**Developer Cheat Sheets (Builder)**

- Authentication Cheat Sheet (Spanish)
- Choosing and Using Security Questions Cheat Sheet
- Clickjacking Defense Cheat Sheet
- C-Based Toolchain Hardening Cheat Sheet

- Cross-Site Request Forgery (CSRF) Prevention Cheat Sheet
- Cryptographic Storage Cheat Sheet
- DOM based XSS Prevention Cheat Sheet
- Forgot Password Cheat Sheet
- HTML5 Security Cheat Sheet
- Input Validation Cheat Sheet
- JAAS Cheat Sheet
- Logging Cheat Sheet
- .NET Security Cheat Sheet
- **OWASP Top Ten Cheat Sheet**
- Password Storage Cheat Sheet
- Pinning Cheat Sheet
- Query Parameterization Cheat Sheet
- Ruby on Rails Cheatsheet
- REST Security Cheat Sheet
- Session Management Cheat Sheet
- SAML Security Cheat Sheet
- SQL Injection Prevention Cheat Sheet
- Transaction Authorization Cheat Sheet
- Transport Layer Protection Cheat Sheet
- Unvalidated Redirects and Forwards Cheat Sheet
- User Privacy Protection Cheat Sheet
- Web Service Security Cheat Sheet
- XSS (Cross Site Scripting) Prevention Cheat Sheet

**Assessment Cheat Sheets (Breaker)**

- Attack Surface Analysis Cheat Sheet
- XSS Filter Evasion Cheat Sheet
- REST Assessment Cheat Sheet
- Web Service Security Testing Cheat Sheet

**Mobile Cheat Sheets**

- IOS Developer Cheat Sheet
- Mobile Jailbreaking Cheat Sheet

**OpSec Cheat Sheets (Defender)**

- Virtual Patching Cheat Sheet

**Draft Cheat Sheets**

- Access Control Cheat Sheet
- Application Security Architecture Cheat Sheet
- Business Logic Security Cheat Sheet
- PHP Security Cheat Sheet
- Secure Coding Cheat Sheet
- Secure SDLC Cheat Sheet
- Threat Modeling Cheat Sheet
- Web Application Security Testing Cheat Sheet
- Grails Secure Code Review Cheat Sheet
- IOS Application Security Testing Cheat Sheet
- Key Management Cheat Sheet
- Insecure Direct Object Reference Prevention Cheat Sheet
- Content Security Policy Cheat Sheet

# Authors and Primary Editors

Andrew van der Stock vanderaj[at]owasp.org

Ismael Rocha Gonçalves ismaelrg[at]gmail.com

Jorge Correa jacorream@gmail.com

# Other Cheatsheets

**Developer Cheat Sheets (Builder)**

- Authentication Cheat Sheet (Spanish)
- Choosing and Using Security Questions Cheat Sheet
- Clickjacking Defense Cheat Sheet
- C-Based Toolchain Hardening Cheat Sheet
- Cross-Site Request Forgery (CSRF) Prevention Cheat Sheet
- Cryptographic Storage Cheat Sheet
- DOM based XSS Prevention Cheat Sheet
- Forgot Password Cheat Sheet
- HTML5 Security Cheat Sheet
- Input Validation Cheat Sheet

- JAAS Cheat Sheet
- Logging Cheat Sheet
- .NET Security Cheat Sheet
- **OWASP Top Ten Cheat Sheet**
- Password Storage Cheat Sheet
- Pinning Cheat Sheet
- Query Parameterization Cheat Sheet
- Ruby on Rails Cheatsheet
- REST Security Cheat Sheet
- Session Management Cheat Sheet
- SAML Security Cheat Sheet
- SQL Injection Prevention Cheat Sheet
- Transaction Authorization Cheat Sheet
- Transport Layer Protection Cheat Sheet
- Unvalidated Redirects and Forwards Cheat Sheet
- User Privacy Protection Cheat Sheet
- Web Service Security Cheat Sheet
- XSS (Cross Site Scripting) Prevention Cheat Sheet

**Assessment Cheat Sheets (Breaker)**

- Attack Surface Analysis Cheat Sheet
- XSS Filter Evasion Cheat Sheet
- REST Assessment Cheat Sheet
- Web Service Security Testing Cheat Sheet

**Mobile Cheat Sheets**

- IOS Developer Cheat Sheet
- Mobile Jailbreaking Cheat Sheet

**OpSec Cheat Sheets (Defender)**

- Virtual Patching Cheat Sheet

**Draft Cheat Sheets**

- Access Control Cheat Sheet
- Application Security Architecture Cheat Sheet
- Business Logic Security Cheat Sheet
- PHP Security Cheat Sheet
- Secure Coding Cheat Sheet
- Secure SDLC Cheat Sheet
- Threat Modeling Cheat Sheet
- Web Application Security Testing Cheat Sheet
- Grails Secure Code Review Cheat Sheet
- IOS Application Security Testing Cheat Sheet
- Key Management Cheat Sheet
- Insecure Direct Object Reference Prevention Cheat Sheet
- Content Security Policy Cheat Sheet