OWASP 2010 Top 10 Cheat Sheet

	Presentation	Controller	Model
A1 Injection	Avoid or prohibit hidden fields, custom headers or cookies *Render* Set a correct content type Set safe character set (UTF-8) Set correct locale *On Submit* Enforce input field type and lengths Validate fields and provide feedback Ensure option selects and radio contain only sent values	**Canonicalize using correct character set** **Positive input validation using correct character set** (NR) Negative input validation (LR) Sanitize input *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** Object relational model (Hibernate) Active Record design pattern Stored procedures Escape mechanisms such as ESAPI's Encoder.EncodeForLDAP() or Encoder.EncodeforOS() *Tip: **All** SQL Injection is due to dynamic SQL queries. Strongly consider prohibiting dynamic SQL queries within your organization*
A2 XSS	Avoid or prohibit hidden fields, custom headers or cookies *Render* **Set correct content type** **Set safe character set (UTF-8)** **Set correct locale** **Output encode all user data as per output context** **Set input constraints** *On Submit* Enforce input field type and lengths Validate fields and provide feedback Ensure option selects and radio contain only sent values	**Canonicalize using correct character set** **Positive input validation using correct character set** (NR) Negative input validation (LR) Sanitize input *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.* *Tip: Use OWASP Scrubbr to clean tainted or hostile data from legacy data*
A3 Weak authentication and session management	*Render* **Validate user is authenticated** **Validate role is sufficient for this view** **Set "secure" and "HttpOnly" flags for session cookies** Send CSRF token with forms	*Design* **Only use inbuilt session management** **Store secondary SSO / framework / custom session identifiers in native session object – do not send as additional headers or cookies** **Validate user is authenticated** **Validate role is sufficient to perform this action** Validate CSRF token	Validate role is sufficient to create, read, update, or delete data *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.*
A4 Insecure Direct Object Reference	**If data is from internal trusted sources, no data is sent** ***Or*** *Render* **Send indirect random access reference map value**	**Obtain data from internal, trusted sources** ***Or*** **Obtain direct value from random access reference access map**	Validate role is sufficient to create, read, update, or delete data
A5 Cross Site Request Forgery	*Pre-render* Validate user is authenticated Validate role is sufficient for this view *Render* **Send CSRF token** Set "secure" and "HttpOnly" flags for session cookies	**Validate CSRF token** Validate role is sufficient to perform this action Validate role is sufficient *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

Legend: NR – Not Recommended :: LR – Last Resort :: **Bold** – primary control mechanism. This cheat sheet is not the only way to achieve compliance with the Top 10 2010.

A6 Security Misconfiguration	Ensure web servers and application servers are hardened PHP – Ensure allow_url_fopen and allow_url_include are both disabled in php.ini. Consider the use of Suhosin extension	Ensure web servers and application servers are hardened 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
A7 Failure to Restrict URL access	*Design* **Ensure all non-web data is outside the web root (logs, configuration, etc)** **Use octet byte streaming instead of providing access to real files such as PDFs or CSVs or similar** **Ensure every page requires a role, even if it is "guest"** *Pre-render* **Validate user is authenticated** **Validate role is sufficient to view secured URL** *Render* **Send CSRF token**	**Validate user is authenticated** **Validate role is sufficient to perform secured action** **Validate CSRF token** *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 the web application server using binary octet streaming.* *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.*	Validate role is sufficient to create, read, update, or delete data
A8 Unvalidated Redirects and Forwards	**Design the app without URL redirection parameters** *or* *Render* Use random indirect object references for redirection parameters	**Design the app without URL redirection parameters** *or* Obtain direct redirection parameter from random indirect reference access map (LR) Positive validation of redirection parameter (NR) Java – Do not forward() requests as this prevents SSO access control mechanisms	Validate role is sufficient to create, read, update, or delete data
A9 Insufficient Cryptographic Storage	*Design* **Use strong ciphers (AES 128 or better)** **Use strong hashes (SHA 256 or better) with salts for passwords** **Protect keys more than any other asset** *Render* Do not send keys or hashes to the browser	*Design* **Use strong ciphers (AES 128 or better)** **Use strong hashes (SHA 256 or better) with salts for passwords** **Protect keys more than any other asset** *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.* *Tip: It is best to encrypt data on the application server, rather than the database server.*	*Design* *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.*
A10 Insufficient Transport Layer Protection	Use TLS 1.2 or later for all web communications Buy extended validation (EV) certificates for public web servers *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.*	Mandate strong encrypted communications between web and database servers and any other servers or administrative users	Mandate strong encrypted communications with application servers and any other servers or administrative users

Legend: NR – Not Recommended :: LR – Last Resort :: **Bold** – primary control mechanism. This cheat sheet is not the only way to achieve compliance with the Top 10 2010.