# OWASP Annotated Application Security Verification Standard Documentation

*Release 3.0.0*

**Boy Baukema**

**Oct 03, 2017**

## Browse by chapter:

CHAPTER 1

# v1 Architecture, design and threat modelling

## 1.1 All components are identified

Verify that all application components are identified and are known to be needed.

Levels: 1, 2, 3

### General

The purpose of this requirement is twofold:

1. Discover third party components that may contain (public) vulnerabilities.
2. Discover 'dead code / dependencies' that increase attack surface without any benefit in functionality.

Developers may be tempted to keep dead code / dependencies around 'in case I ever need it', however this is not an argument for a shippable product, such dead code / dependencies is better left on a source control system.

## 1.2 All dependencies are identified

Verify that all components, such as libraries, modules, and external systems, that are not part of the application but that the application relies on to operate are identified.

Levels: 2, 3

## 1.3 A high-level architecture as been defined

Verify that a high-level architecture for the application has been defined.

Levels: 2, 3

## 1.4 All components are defined

Verify that all application components are defined in terms of the business functions and/or security functions they provide.

Levels: 3

## 1.5 All dependant components are defined

Verify that all components that are not part of the application but that the application relies on to operate are defined in terms of the functions, and/or security functions, they provide.

Levels: 3

## 1.6 A STRIDE threat model has been produced

Verify that a threat model for the target application has been produced and covers off risks associated with Spoofing, Tampering, Repudiation, Information Disclosure, and Elevation of privilege (STRIDE).

Levels: 3

## 1.7 Central implementation for security controls

Verify all security controls (including libraries that call external security services) have a centralized implementation.

Levels: 3

## 1.8 Components are segregated

Verify that components are segregated from each other via a defined security control, such as network segmentation, firewall rules, or cloud based security groups.

Levels: 2, 3

## 1.9 Clear separation between data, controller and display

Verify the application has a clear separation between the data layer, controller layer and the display layer, such that security decisions can be enforced on trusted systems.

Levels: 2, 3

## 1.10 Client side logic does not contain secrets

Verify that there is no sensitive business logic, secret keys or other proprietary information in client side code.

Levels: 2, 3

CHAPTER 2

# v2 Authentication verification requirements

## 2.1 Principle of complete mediation

Verify all pages and resources by default require authentication except those specifically intended to be public (Principle of complete mediation).

Levels: 1, 2, 3

### Drupal 7

This verification is highly likely to fail, though it may have a LOW risk. Tests these things:

#### Publicly accessible files

By default Drupal puts everything in the webroot, so a lot of static assets are available. Among these are all the module and library files, look through all of these and look for .html or .php files that could be used to do XSS, Remote File Inclusion, Remote Code Execution, etc.

#### 'access callback' => TRUE

Drupal routing relies on hook_menu. Check instances of this hook, especially in custom modules, for 'access callback' => TRUE, meaning that this functionality is public.

### General

Basically (for a normal web app) this means checking:

- There is some sort of default rule what specifies that unless explicitly whitelisted.
- The whitelisting rules are not overly broad.

The nice thing about the use of 'all' here is that you only need to find a single thing that probably shouldn't be public to FAIL this rule.

Unfortunately though, this is something that relies heavily on the auditors judgement:

- pages and resources (is a database a resource?)
- authentication (is appending debug=true to a URL authentication? or obfuscated URLs that may or may not be brute force guessable?)
- intended to be public (99.999% of the intentions for the application available pages/resources are probably not specified anywhere)

Feel free to contact the customer / Team Lead if uncertain about any of these terms in the context of the application.

### PHP

Check the following:

- Is there some prescribed set of rewriterules for the webserver (Apache/Nginx) or a .htaccess file?
- What files can be found in the webroot? Should all those files be publicly accessible?
- How is routing configured? If MVC is used, were you to add a new action, could you immediately call it via a URL?

### Symfony 2

The Controller verification depends entirely on the firewall configuration in `app/config/security.yml`. This should look something like:

So by default all URLs requested should be authenticated, except those explicitly whitelisted. Check that the whilelisted regular expressions are not.

**Watch out** Beware for the anonymous key though: if the anonymous key is set, all URLs are accessible through the firewall for anonymous users, they are then guarded by the access_control settings section. Therefor, if the anonymous key is used, all routes that the firewall guards MUST have roles defined that are allowed to access that URL. More on this can be found in the Symfony Documentation regarding the firewall. Don't forget that the application also has a CLI interface. Usually this should only be available to (pre-authenticated via SSH) administrators, if so, mention this like so:

> The TOV also supports several administrative command line commands, for security these rely on the administrator to authenticate via SSH (out of scope of this verification).

## 2.2 Password fields

Verify that all password fields do not echo the user's password when it is entered.

Levels: 1, 2, 3

### Drupal 7

This verification is highly likely to fail, though it may have a LOW risk. Tests these things:

### Publicly accessible files

By default Drupal puts everything in the webroot, so a lot of static assets are available. Among these are all the module and library files, look through all of these and look for .html or .php files that could be used to do XSS, Remote File Inclusion, Remote Code Execution, etc.

### 'access callback' => TRUE

Drupal routing relies on hook_menu. Check instances of this hook, especially in custom modules, for 'access callback' => TRUE, meaning that this functionality is public.

### General

Basically (for a normal web app) this means checking:

- There is some sort of default rule what specifies that unless explicitly whitelisted.
- The whitelisting rules are not overly broad.

The nice thing about the use of 'all' here is that you only need to find a single thing that probably shouldn't be public to FAIL this rule.

Unfortunately though, this is something that relies heavily on the auditors judgement:

- pages and resources (is a database a resource?)
- authentication (is appending debug=true to a URL authentication? or obfuscated URLs that may or may not be brute force guessable?)
- intended to be public (99.999% of the intentions for the application available pages/resources are probably not specified anywhere)

Feel free to contact the customer / Team Lead if uncertain about any of these terms in the context of the application.

### PHP

Check the following:

- Is there some prescribed set of rewriterules for the webserver (Apache/Nginx) or a .htaccess file?
- What files can be found in the webroot? Should all those files be publicly accessible?
- How is routing configured? If MVC is used, were you to add a new action, could you immediately call it via a URL?

### Symfony 2

The Controller verification depends entirely on the firewall configuration in `app/config/security.yml`. This should look something like:

So by default all URLs requested should be authenticated, except those explicitly whitelisted. Check that the whilelisted regular expressions are not.

**Watch out** Beware for the anonymous key though: if the anonymous key is set, all URLs are accessible through the firewall for anonymous users, they are then guarded by the access_control settings section. Therefor, if the anonymous key is used, all routes that the firewall guards MUST have roles defined that are allowed to access that URL. More on this can be found in the Symfony Documentation regarding the firewall. Don't forget that the application also has a

CLI interface. Usually this should only be available to (pre-authenticated via SSH) administrators, if so, mention this like so:

> The TOV also supports several administrative command line commands, for security these rely on the administrator to authenticate via SSH (out of scope of this verification).

## 2.4 Server side enforcement

Verify all authentication controls are enforced on the server side.

Levels: 1, 2, 3

### General

If the client uses a back-end API (say over an XMLHttpRequest) then the actual API has to check that the client has previously authenticated (by expecting some form of HTTP or other authentication). Note that if the application fails this validation it will almost certainly also fail V2.1.

## 2.6 Fails securely

Verify all authentication controls fail securely to ensure attackers cannot log in.

Levels: 1, 2, 3

### Drupal 7

The default Drupal authentication controls, if used, pass this requirement.

### General

Check the error handling on authentication controls (check for badly implemented authentication, most frameworks will solve this for a developer). Is it possible to trigger an error that disables authentication entirely? If you can, take the database offline and see if the application will allow you to log in. Or see if there is a dependency on some external service provider.

### Symfony 2

The default Symfony2 authentication controls, if used, pass this requirement.

## 2.7 Allows for strong passwords

Verify password entry fields allow, or encourage, the use of passphrases, and do not prevent long passphrases/highly complex passwords being entered.

Levels: 1, 2, 3

## 2.8 All account identity authentication functions are secure

Verify all account identity authentication functions (such as update profile, forgot password, disabled / lost token, help desk or IVR) that might regain access to the account are at least as resistant to attack as the primary authentication mechanism.

Levels: 1, 2, 3

## 2.9 All credential changes are secure

Verify that the changing password functionality includes the old password, the new password, and a password confirmation.

Levels: 1, 2, 3

## 2.12 All authentication decisions are logged

Verify that all suspicious authentication decisions are logged. This should include requests with relevant metadata needed for security investigations.

Levels: 2, 3

## 2.13 Account passwords are salted properly

Verify that account passwords make use of a sufficient strength encryption routine and that it withstands brute force attack against the encryption routine.

Levels: 2, 3

## 2.16 Strongly encrypted transport

Verify that credentials are transported using a suitable encrypted link and that all pages/functions that require a user to enter credentials are done so using an encrypted link.

Levels: 1, 2, 3

### Drupal 7

Check /user. But don't forget that the installation may host other credentials / PII information.

### General

To verify this you will most likely need a deployed application, the easiest way to verify this is to navigate with an up to date browser to the login page. The login page and all subsequent authenticated pages must be exclusively accessed over TLS. Also try to force the login page to load over HTTP, this should fail. HTTP Strict Transport Security should be used. Use the SSL Server Test by Qualys to verify that the TLS has been correctly implemented. Slightly less egregious than sending credentials over plain HTTP, but still a risk to the user, check that Personal Identifiable Information is not sent over plain HTTP (where anyone eaves dropping may pick it up).

## 2.17 No clear text passwords

Verify that the forgotten password function and other recovery paths do not reveal the current password and that the new password is not sent in clear text to the user.

Levels: 1, 2, 3

### Drupal 7

Default Drupal authentication, if used, meets this requirement.

### General

Try the 'password forgotten' functionality if it exists and see if you get sent your own password back in cleartext. Note that one-time-passwords or tokens are okay, as long as they expire upon use.

## 2.18 No username enumeration

Verify that information enumeration is not possible via login, password reset, or forgot account functionality.

Levels: 1, 2, 3

### Drupal 7

By default Drupal does not meet this requirement, see also: Drupal.org Project: username Enumeration Prevention.

### General

A system should not disclose whether a username (usually half of the credentials required to log in) is valid or not. Note that this may actually cary a business risk, that is that if I disclose that someone is a member a competitor may use this as an advantage to try to 'steal' the customer. Or it might put the user in an uncomfortable position, if an attacker can verify that he/she is a member of certain sites. A simple test can be to try to authenticate first with a non-existent account. Note that in theory you might also use a timing attack for this. Non-existent usernames may give an error page faster than usernames that are known but where the password supplied is wrong, especially with a hashing mechanism like bcrypt.

### Symfony 2

The default authentication controls for Symfony2 meet this requirement.

## 2.19 No default passwords

Verify there are no default passwords in use for the application framework or any components used by the application (such as "admin/password").

Levels: 1, 2, 3

### Drupal 7

Drupal has a default user #1, often named 'admin' or 'administrator'. See also: (Drupal.org: Securing user #1)[https://drupal.org/node/947312]

### General

For inspiration you can check out lists like those found on: defaultpassword.com.

## 2.20 Protects against brute force attacks

Verify that request throttling is in place to prevent automated attacks against common authentication attacks such as brute force attacks or denial of service attacks.

Levels: 1, 2, 3

## 2.21 External service credentials are encrypted and protected

Verify that all authentication credentials for accessing services external to the application are encrypted and stored in a protected location.

Levels: 2, 3

## 2.22 Password recovery is well implemented

Verify that forgotten password and other recovery paths use a soft token, mobile push, or an offline recovery mechanism.

Levels: 1, 2, 3

## 2.23 Password recovery can not be used to lock out users

Verify that account lockout is divided into soft and hard lock status, and these are not mutually exclusive. If an account is temporarily soft locked out due to a brute force attack, this should not reset the hard lock status.

Levels: 2, 3

## 2.24 No "secret" questions

Verify that if knowledge based questions (also known as "secret questions") are required, the questions should be strong enough to protect the application.

Levels: 1, 2, 3

## 2.25 Supports configuration to disallow previous passwords

Verify that the system can be configured to disallow the use of a configurable number of previous passwords.

Levels: 2, 3

## 2.26 Sensitive operations are sufficiently protected

Verify re-authentication, step up or adaptive authentication, two factor authentication, or transaction signing is required before any application-specific sensitive operations are permitted as per the risk profile of the application.

Levels: 2, 3

## 2.27 Block common or weak passwords / passphrases

Verify that measures are in place to block the use of commonly chosen passwords and weak passphrases.

Levels: 1, 2, 3

## 2.28 Authentication success or failure should take equal time

Verify that all authentication challenges, whether successful or failed, should respond in the same average response time.

Levels: 3

## 2.29 Secrets are not included in the source code

Verify that secrets, API keys, and passwords are not included in the source code, or online source code repositories.

Levels: 3

## 2.30 Use a proven secure authentication mechanism

Verify that if an application allows users to authenticate, they use a proven secure authentication mechanism.

Levels: 1, 2, 3

## 2.31 Protects against username & password disclosure

Verify that if an application allows users to authenticate, they can authenticate using two-factor authentication or other strong authentication, or any similar scheme that provides protection against username + password disclosure.

Levels: 2, 3

## 2.32 Admin is not accessible for untrusted parties

Verify that administrative interfaces are not accessible to untrusted parties

Levels: 1, 2, 3

CHAPTER 3

# v3 Session management verification requirements

## 3.1 Uses default session management

Verify that there is no custom session manager, or that the custom session manager is resistant against all common session management attacks.

Levels: 1, 2, 3

### General

> Implementations built from scratch are often weak and breakable. Developers are strongly discouraged from implementing their own Session Management. Leading web frameworks have undergone rounds of testing and fixing that leave them using secure methods of token generation. There is no value in re-writing such basic building blocks.

- OWASP: Session Management: Use Only the Framework's Session Manager

### PHP

First grep the codebase for usage of 'session_set_save_handler'. Though (rarely) it's also possible that a developer could have implemented his own session mechanism without even using the default session functions. For this you'll have to look at each part of the application and how it stores and retrieves state across page requests.

## 3.2 Sessions are invalidated on user log out

Verify that sessions are invalidated when the user logs out.

Levels: 1, 2, 3

### Drupal 7

Drupal does this by default for /user/logout, but be wary of custom frontend logins!

### General

If a session can still be used after logging out then the lifetime of the session is increased and that gives third parties that may have intercepted the session token more (or perhaps infinite, if no absolute session expiry happens) time to impersonate a user.

One quick way to test this is to log in, get the session token from the cookie, log out, then manually add the session cookie with the session token and see if you are still logged in.

### PHP

Look for session_destroy in the logout functionality.

### Symfony 2

This is a setting in security.yml:

used by the default session controls.

## 3.3 Session times out after inactivity

Verify that sessions timeout after a specified period of inactivity.

Levels: 1, 2, 3

### Drupal 7

Default session controls do not pass this requirement. See Drupal.org: Session Expire project.

### General

> All sessions should implement an idle or inactivity timeout. This timeout defines the amount of time a session will remain active in case there is no activity in the session, closing and invalidating the session upon the defined idle period since the last HTTP request received by the web application for a given session ID. The idle timeout limits the chances an attacker has to guess and use a valid session ID from another user. However, if the attacker is able to hijack a given session, the idle timeout does not limit the attacker's actions, as he can generate activity on the session periodically to keep the session active for longer periods of time. Session timeout management and expiration must be enforced server-side. If the client is used to enforce the session timeout, for example using the session token or other client parameters to track time references (e.g. number of minutes since login time), an attacker could manipulate these to extend the session duration.

- OWASP Session Management Cheat Sheet: Idle Timeout

### PHP

Default session controls do not pass this requirement. See StackOverflow: How do I expire a PHP session after 30 minutes?.

### Symfony 2

Default session controls do not pass this requirement.

## 3.4 Session has absolute timeout

Verify that sessions timeout after an administratively-configurable maximum time period regardless of activity (an absolute timeout).

Levels: 3

## 3.5 Shows logout link

Verify that all pages that require authentication have easy and visible access to logout functionality.

Levels: 1, 2, 3

### General

> Check that the application provides a logout button and that this button is present and well visible on all pages that require authentication. A logout button that is not clearly visible, or that is present only on certain pages, poses a security risk, as the user might forget to use it at the end of his/her session.

- OWASP: Testing for Logout and Browser Cache Management (OWASP-AT-007)

Note that for larger applications it may be difficult to test all pages, try to find different areas or application flows of the application and check each one briefly.

## 3.6 Does not disclose session id

Verify that the session id is never disclosed in URLs, error messages, or logs. This includes verifying that the application does not support URL rewriting of session cookies.

Levels: 1, 2, 3

### Drupal 7

Drupals bootstrapping sets the session.user_only_cookies value to 1, thereby passing this requirement.

### General

Session IDs in URLs lead to all kinds of bad behaviour (logging by servers, but also accidental sharing by users either manually "here look at this!" or through their browser history).

You can test whether the application allows session ids in the URL by taking a session identifier from one a cookie in one browser session and adding it to the URL query string in another browser session.

Session Ids in error messages and logs may be undesirable depending on who has access to logs.

### PHP

This relies on the following php.ini setting:

```
; This option forces PHP to fetch and use a cookie for storing and maintaining
; the session id. We encourage this operation as it's very helpful in combating
; session hijacking when not specifying and managing your own session id. It is
; not the end all be all of session hijacking defence, but it's a good start.
; http://php.net/session.use-only-cookies
session.use_only_cookies = 1
```

By default this is set to 1, however it is advisable to not rely on the environment for this but set it manually using ini_set.

### Symfony 2

By default this configuration value is not set, but can be set with:

## 3.7 Session id is changed on login

Verify that all successful authentication and re-authentication generates a new session and session id.

Levels: 1, 2, 3

## 3.10 Session ids may only come from framework

Verify that only session ids generated by the application framework are recognized as active by the application.

Levels: 2, 3

## 3.11 Session tokens are sufficiently long and random

Verify that session ids are sufficiently long, random and unique across the correct active session base.

Levels: 1, 2, 3

## 3.12 Session cookies have appropriately restricted paths

Verify that session ids stored in cookies have their path set to an appropriately restrictive value for the application, and authentication session tokens additionally set the "HttpOnly" and "secure" attributes

Levels: 1, 2, 3

## 3.16 Does not permit duplicate concurrent user sessions from different machines

Verify that the application limits the number of active concurrent sessions.

Levels: 1, 2, 3

## 3.17 User can see and terminate all his sessions

Verify that an active session list is displayed in the account profile or similar of each user. The user should be able to terminate any active session.

Levels: 1, 2, 3

## 3.18 User is prompted for session termination on password change

Verify the user is prompted with the option to terminate all other active sessions after a successful change password process.

Levels: 1, 2, 3

CHAPTER 4

# v4 Access control verification requirements

## 4.1 Authorisation of functions and services

Verify that the principle of least privilege exists - users should only be able to access functions, data files, URLs, controllers, services, and other resources, for which they possess specific authorization. This implies protection against spoofing and elevation of privilege.

Levels: 1, 2, 3

### General

This terminology seems to come from IBM Cognos:

> Examples of the secured functions are Administration and Report Studio. Examples of the secured features are User Defined SQL and Bursting IBM® Cognos® 8 Administration and Security Guide 8.4.0: Secured Functions and Features

A 'function' could be described as a separate piece of functionality, for instance for Google Apps this would be 'Google Calendar' or 'Google Mail'. A 'feature' could be described as an action that can be taken, for instance "Add an event from an e-mail to my calendar". Both are slices of functionality to which rights have been applied.

Take the example of a news site with 3 roles: 1. (Anonymous) Reader 2. Editor 3. Editor-in-chief

And a system with 3 'functions': * News front-end, (all roles) ** feature: 'Read news', (all roles) * News administration, (editor and editor-in-chief) ** feature: 'Add news', (editor and editor-in-chief) ** feature: 'Publish news' (editor-in-chief)

To pass this requirement you should verify: * That a Reader may not access the News administration and may not add or publish news. * That an Editor may access all functions and may read news, add news, but not publish news. * That an Editor-in-chief may exercise all aforementioned functions and features.

While this is doable for such a simple example, this matrix can quickly become very large. Remember though that security testing does not require you to test access that should succeed, but instead it requires that you focus on what should NOT happen.

Should you need to reduce the amount of time you can spend verifying this you can do one of the following: 1. Use the source code to extrapolate based on security controls. For example: Both Add News and Publish News are Controller actions protected by the Symfony2 @Secure annotation from anonymous access. After testing Add News as an Anonymous Reader, it can be said with reasonable certainty that Publish News gives the same results. 2. Threat Modelling (OWASP: Threat Risk Modeling) may help reduce the time required by specifying that an Editor Publishing news, may be less of a concern than a Reader adding and/or publishing news.

Note though that doing either will mean that this requirement is not passed, but it can be used to reduce the associated risk of not meeting this requirement.

## 4.4 Authorisation of direct object references

Verify that access to sensitive records is protected, such that only authorized objects or data is accessible to each user (for example, protect against users tampering with a parameter to see or alter another user's account).

Levels: 1, 2, 3

### General

> A direct object reference occurs when a developer exposes a reference to an internal implementation object, such as a file, directory, database record, or key, as a URL or form parameter. An attacker can manipulate direct object references to access other objects without authorization, unless an access control check is in place. For example, in Internet Banking applications, it is common to use the account number as the primary key. Therefore, it is tempting to use the account number directly in the web interface. Even if the developers have used parameterized SQL queries to prevent SQL injection, if there is no extra check that the user is the account holder and authorized to see the account, an attacker tampering with the account number parameter can see or change all accounts. This type of attack occurred to the Australian Taxation Office's GST Start Up Assistance site in 2000, where a legitimate but hostile user simply changed the ABN (a company tax id) present in the URL. The user farmed around 17,000 company details from the system, and then e-mailed each system. This was a major embarrassment to the Government of the 17,000 companies with details of his attack. This type of vulnerability is very common, but is largely untested in current applications.

- OWASP: Top 10 2007-Insecure Direct Object Reference

## 4.5 Disabled directory browsing

Verify that directory browsing is disabled unless deliberately desired. Additionally, applications should not allow discovery or disclosure of file or directory metadata, such as Thumbs.db, .DS_Store, .git or .svn folders.

Levels: 1, 2, 3

### Apache 2

See: Apache wiki: DirectoryListings and documentation for mod_autoindex. Note that the application may have a .htaccess file instructing the webserver to turn on or of 'Indexes'.

### General

This is typically a webserver feature concern (Apache, IIS, Nginx, etc.) that may be on by default and should be turned off.

## 4.8 Access controls fail securely

Verify that access controls fail securely.

Levels: 1, 2, 3

## 4.9 Access control rules are enfoced server side

Verify that the same access control rules implied by the presentation layer are enforced on the server side.

Levels: 1, 2, 3

## 4.10 User and data attributes and policy information cannot be manipulated unauthorized

Verify that all user and data attributes and policy information used by access controls cannot be manipulated by end users unless specifically authorized.

Levels: 2, 3

## 4.11 Access controls are enforced on the server side

Verify that there is a centralized mechanism (including libraries that call external authorization services) for protecting access to each type of protected resource.

Levels: 3

## 4.12 Has centralized mechanism for access to protected resources

Verify that all access control decisions can be logged and all failed decisions are logged.

Levels: 2, 3

## 4.13 Protects against CSRF

Verify that the application or framework uses strong random anti-CSRF tokens or has another transaction protection mechanism.

Levels: 1, 2, 3

## 4.14 Access control decisions and failed decisions are logged

Verify the system can protect against aggregate or continuous access of secured functions, resources, or data. For example, consider the use of a resource governor to limit the number of edits per hour or to prevent the entire database from being scraped by an individual user.

Levels: 2, 3

## 4.15 Protects against fraud

Verify the application has additional authorization (such as step up or adaptive authentication) for lower value systems, and / or segregation of duties for high value applications to enforce anti-fraud controls as per the risk of application and past fraud.

Levels: 2, 3

## 4.16 Protects against parameter tampering

Verify that the application correctly enforces context-sensitive authorisation so as to not allow unauthorised manipulation by means of parameter tampering.

Levels: 1, 2, 3

CHAPTER 5

# v5 Malicious input handling verification requirements

## 5.1 Buffer overflows

Verify that the runtime environment is not susceptible to buffer overflows, or that security controls prevent buffer overflows.

Levels: 1, 2, 3

### General

> A buffer overflow condition exists when a program attempts to put more data in a buffer than it can hold or when a program attempts to put data in a memory area past a buffer. In this case, a buffer is a sequential section of memory allocated to contain anything from a character string to an array of integers. Writing outside the bounds of a block of allocated memory can corrupt data, crash the program, or cause the execution of malicious code. * OWASP: Buffer Overflow

### PHP

Applications written in PHP are not vulnerable to Buffer Overflows.

## 5.3 Rejects invalid input

Verify that server side input validation failures result in request rejection and are logged.

Levels: 1, 2, 3

## 5.5 Input validation or encoding is performed and enforced on the server side.

Verify that input validation routines are enforced on the server side.

Levels: 1, 2, 3

## 5.6 One input validation control per type of accepted data

Verify that a single input validation control is used by the application for each type of data that is accepted.

Levels: 3

## 5.10 SQL Injection

Verify that all SQL queries, HQL, OSQL, NOSQL and stored procedures, calling of stored procedures are protected by the use of prepared statements or query parameterization, and thus not susceptible to SQL injection

Levels: 1, 2, 3

## 5.11 LDAP Injection

Verify that the application is not susceptible to LDAP Injection, or that security controls prevent LDAP Injection.

Levels: 1, 2, 3

## 5.12 OS Command Injection

Verify that the application is not susceptible to OS Command Injection, or that security controls prevent OS Command Injection.

Levels: 1, 2, 3

## 5.13 XXE

Verify that the application is not susceptible to Remote File Inclusion (RFI) or Local File Inclusion (LFI) when content is used that is a path to a file.

Levels: 1, 2, 3

## 5.14 XML Injection

Verify that the application is not susceptible to common XML attacks, such as XPath query tampering, XML External Entity attacks, and XML injection attacks.

Levels: 1, 2, 3

## 5.15 TODO

Ensure that all string variables placed into HTML or other web client code is either properly contextually encoded manually, or utilize templates that automatically encode contextually to ensure the application is not susceptible to reflected, stored and DOM Cross-Site Scripting (XSS) attacks.

Levels: 1, 2, 3

## 5.16 HTML escaping

If the application framework allows automatic mass parameter assignment (also called automatic variable binding) from the inbound request to a model, verify that security sensitive fields such as "accountBalance", "role" or "password" are protected from malicious automatic binding.

Levels: 2, 3

## 5.17 Protected against malicious automatic binding

Verify that the application has defenses against HTTP parameter pollution attacks, particularly if the application framework makes no distinction about the source of request parameters (GET, POST, cookies, headers, environment, etc.)

Levels: 2, 3

## 5.18 Defends against HTTP parameter pollution attacks

Verify that client side validation is used as a second line of defense, in addition to server side validation.

Levels: 2, 3

## 5.19 Output encoding/escaping has a single security control per type

Verify that all input data is validated, not only HTML form fields but all sources of input such as REST calls, query parameters, HTTP headers, cookies, batch files, RSS feeds, etc; using positive validation (whitelisting), then lesser forms of validation such as greylisting (eliminating known bad strings), or rejecting bad inputs (blacklisting).

Levels: 2, 3

## 5.20 Structured data is strongly typed and validated with a schema

Verify that structured data is strongly typed and validated against a defined schema including allowed characters, length and pattern (e.g. credit card numbers or telephone, or validating that two related fields are reasonable, such as validating suburbs and zip or post codes match).

Levels: 2, 3

## 5.21 Unstructured data is sanitized

Verify that unstructured data is sanitized to enforce generic safety measures such as allowed characters and length, and characters potentially harmful in given context should be escaped (e.g. natural names with Unicode or apostrophes, such as  or O'Hara)

Levels: 2, 3

## 5.22 Untrusted HTML is sanitized

Make sure untrusted HTML from WYSIWYG editors or similar are properly sanitized with an HTML sanitizer and handle it appropriately according to the input validation task and encoding task.

Levels: 1, 2, 3

## 5.23 Auto escaping technology always applies HTML sanitization

For auto-escaping template technology, if UI escaping is disabled, ensure that HTML sanitization is enabled instead.

Levels: 2, 3

## 5.24 DOM writes use safe JavaScript methods

Verify that data transferred from one DOM context to another, uses safe JavaScript methods, such as using .innerText and .val.

Levels: 2, 3

## 5.25 JSON is properly parsed by browser

Verify when parsing JSON in browsers, that JSON.parse is used to parse JSON on the client. Do not use eval() to parse JSON on the client.

Levels: 2, 3

## 5.26 Data is cleared from client storage on session termination

Verify that authenticated data is cleared from client storage, such as the browser DOM, after the session is terminated.

Levels: 2, 3

CHAPTER 6

# v6 Output encoding / escaping

CHAPTER 7

# v7 Cryptography at rest verification requirements

## 7.2 Crypto fails securely

Verify that all cryptographic modules fail securely, and errors are handled in a way that does not enable oracle padding.

Levels: 1, 2, 3

## 7.6 Random numbers, file names, GUIDs and strings are sufficiently random

Verify that all random numbers, random file names, random GUIDs, and random strings are generated using the cryptographic module's approved random number generator when these random values are intended to be not guessable by an attacker.

Levels: 2, 3

## 7.7 Crypto modules have been validated against FIPS 140-2 or equivalent

Verify that cryptographic algorithms used by the application have been validated against FIPS 140-2 or an equivalent standard.

Levels: 1, 2, 3

## 7.8 Crypto modules operate in their approved mode

Verify that cryptographic modules operate in their approved mode according to their published security policies.

Levels: 3

## 7.9 Policy for cryptographic key management exists and is enforced

Verify that there is an explicit policy for how cryptographic keys are managed (e.g., generated, distributed, revoked, and expired). Verify that this key lifecycle is properly enforced.

Levels: 2, 3

## 7.11 Cryptographic processes are isolated

Verify that all consumers of cryptographic services do not have direct access to key material. Isolate cryptographic processes, including master secrets and consider the use of a hardware key vault (HSM).

Levels: 3

## 7.12 PII is encrypted at rest and protected during communication

Personally Identifiable Information should be stored encrypted at rest and ensure that communication goes via protected channels.

Levels: 2, 3

## 7.13 Keys and secrets are zeroed when destroyed

Verify that where possible, keys and secrets are zeroed when destroyed.

Levels: 2, 3

## 7.14 Secrets are replaceable and placed at installation

Verify that all keys and passwords are replaceable, and are generated or replaced at installation time.

Levels: 2, 3

## 7.15 Random numbers are sufficiently random even under load

Verify that random numbers are created with proper entropy even when the application is under heavy load, or that the application degrades gracefully in such circumstances.

Levels: 3

CHAPTER 8

# v8 Error handling and logging verification requirements

## 8.1 Information leakage

Verify that the application does not output error messages or stack traces containing sensitive data that could assist an attacker, including session id, software/framework versions and personal information

Levels: 1, 2, 3

## 8.2 Error handling is performed on trusted devices

Verify that error handling logic in security controls denies access by default.

Levels: 2, 3

## 8.3 Logging controls are implemented on the server

Verify security logging controls provide the ability to log success and particularly failure events that are identified as security-relevant.

Levels: 2, 3

## 8.4 Error handling logic denies access by default

Verify that each log event includes necessary information that would allow for a detailed investigation of the timeline when an event happens.

Levels: 2, 3

## 8.5 Security relevant success and failure events are loggable by controls

Verify that all events that include untrusted data will not execute as code in the intended log viewing software.

Levels: 3

## 8.6 Log events are complete

Verify that security logs are protected from unauthorized access and modification.

Levels: 2, 3

## 8.7 Events that include untrusted data will not be executed

Verify that the application does not log sensitive data as defined under local privacy laws or regulations, organizational sensitive data as defined by a risk assessment, or sensitive authentication data that could assist an attacker, including user's session identifiers, passwords, hashes, or API tokens.

Levels: 2, 3

## 8.8 Security logs are protected

Verify that all non-printable symbols and field separators are properly encoded in log entries, to prevent log injection.

Levels: 3

## 8.9 Single application-level logging implementation

Verify that log fields from trusted and untrusted sources are distinguishable in log entries.

Levels: 3

## 8.10 Application log does not include sensitive data

Verify that an audit log or similar allows for non-repudiation of key transactions.

Levels: 2, 3

## 8.11 A sufficiently advanced log analysis tool is available

Verify that security logs have some form of integrity checking or controls to prevent unauthorized modification.

Levels: 3

## 8.12 Logs are stored differently and rotated

Verify that the logs are stored on a different partition than the application is running with proper log rotation.

Levels: 3

CHAPTER 9

# v9 Data protection verification requirements

## 9.1 Sensitive data does not get cached

Verify that all forms containing sensitive information have disabled client side caching, including autocomplete features.

Levels: 1, 2, 3

### General

Pre-filled forms may be cached by the browser (even when using HTTPS), check caching headers.

Turning off autocomplete is very debatable and probably pretty LOW risk.

Look for all places where sensitive information (Personal Identifiable Information, Credit Card info, secrets / tokens) is entered and make sure that autocomplete is explicitly disabled. If not, then step back and think about the risk of autocomplete by asking the following questions: * How likely is it that this application will be or is being used in a shared environment (like a internet cafe)? If very likely then the risk might be MEDIUM. * How likely is it very old browsers (like IE 6/7 or Safari 4/5) will be using this application? If very likely then the risk might be MEDIUM. Otherwise the risk will probably be LOW.

But be careful, when chained with XSS this vulnerability can make it possible for an attacker to steal sensitive information without the user actively entering it.

## 9.2 Sensitive data is identified and access policy exists and is enforced

Verify that the list of sensitive data processed by the application is identified, and that there is an explicit policy for how access to this data must be controlled, encrypted and enforced under relevant data protection directives.

Levels: 3

## 9.3 Sensitive data does not get sent in the URL

Verify that all sensitive data is sent to the server in the HTTP message body or headers (i.e., URL parameters are never used to send sensitive data).

Levels: 1, 2, 3

### General

Sending sensitive data (like Personable Identifiable Information, Credit Card numbers, passwords / tokens) in the URL will lead to them being available in the browser history and in the logs by the application server and any potential intermediaries (proxies).

## 9.4 Temporary client caches of sensitive data are properly cleaned up

Verify that the application sets appropriate anti-caching headers as per the risk of the application, such as the following: Expires: Tue, 03 Jul 2001 06:00:00 GMT Last-Modified: {now} GMT Cache-Control: no-store, no-cache, must-revalidate, max-age=0 Cache-Control: post-check=0, pre-check=0 Pragma: no-cache

Levels: 1, 2, 3

## 9.5 Temporary server caches of sensitive data are properly cleaned up

Verify that on the server, all cached or temporary copies of sensitive data stored are protected from unauthorized access or purged/invalidated after the authorized user accesses the sensitive data.

Levels: 2, 3

## 9.6 Sensitive data can be removed after required retention period

Verify that there is a method to remove each type of sensitive data from the application at the end of the required retention policy.

Levels: 3

## 9.7 Minimal parameters are sent to untrusted systems

Verify the application minimizes the number of parameters in a request, such as hidden fields, Ajax variables, cookies and header values.

Levels: 2, 3

## 9.8 Abnormal behaviour is detectable

Verify the application has the ability to detect and alert on abnormal numbers of requests for data harvesting for an example screen scraping.

Levels: 3

## 9.9 Client side storage does not contain secrets

Verify that data stored in client side storage - such as HTML5 local storage, session storage, IndexedDB, regular cookies or Flash cookies - does not contain sensitive or PII).

Levels: 1, 2, 3

## 9.10 Accessing sensitive data is logged

Verify accessing sensitive data is logged, if the data is collected under relevant data protection directives or where logging of accesses is required.

Levels: 2, 3

## 9.11 Sensitive data is rapidly sanitized from memory

Verify that sensitive data is rapidly sanitized from memory as soon as it is no longer needed and handled in accordance to functions and techniques supported by the framework/library/operating system.

Levels: 2, 3

CHAPTER 10

# v10 Communications security verification requirements

## 10.1 TLS chain is valid

Verify that a path can be built from a trusted CA to each Transport Layer Security (TLS) server certificate, and that each server certificate is valid.

Levels: 1, 2, 3

### General

Simplest verification for this is to let the browser do the work and visit the application with a modern browser.

## 10.3 TLS is used for all relevant connections

Verify that TLS is used for all connections (including both external and backend connections) that are authenticated or that involve sensitive data or functions, and does not fall back to insecure or unencrypted protocols. Ensure the strongest alternative is the preferred algorithm.

Levels: 1, 2, 3

## 10.4 Backend TLS connection failures are logged

Verify that backend TLS connection failures are logged.

Levels: 3

## 10.5 Client certificates are built and verified correctly

Verify that certificate paths are built and verified for all client certificates using configured trust anchors and revocation information.

Levels: 3

## 10.6 Connections to relevant external systems are authenticated

Verify that all connections to external systems that involve sensitive information or functions are authenticated.

Levels: 2, 3

## 10.8 Single standard well-configured TLS implementation is used

Verify that there is a single standard TLS implementation that is used by the application that is configured to operate in an approved mode of operation.

Levels: 3

## 10.10 Certificate pinning is used correctly

Verify that TLS certificate public key pinning is implemented with production and backup public keys. For more information, please see the references below.

Levels: 3

## 10.11 Strict Transport Security is used correctly

Verify that HTTP Strict Transport Security headers are included on all requests and for all subdomains, such as Strict-Transport-Security: max-age=15724800; includeSubdomains

Levels: 1, 2, 3

## 10.12 URL is submitted to HSTS preload lists

Verify that production website URL has been submitted to preloaded list of Strict Transport Security domains maintained by web browser vendors. Please see the references below.

Levels: 3

## 10.13 Forward secrecy ciphers are used

Ensure forward secrecy ciphers are in use to mitigate passive attackers recording traffic.

Levels: 1, 2, 3

## 10.14 Certification revocation is enabled and configured

Verify that proper certification revocation, such as Online Certificate Status Protocol (OSCP) Stapling, is enabled and configured.

Levels: 1, 2, 3

## 10.15 Strong certificate hierarchy

Verify that only strong algorithms, ciphers, and protocols are used, through all the certificate hierarchy, including root and intermediary certificates of your selected certifying authority.

Levels: 1, 2, 3

## 10.16 TLS settings are current

Verify that the TLS settings are in line with current leading practice, particularly as common configurations, ciphers, and algorithms become insecure.

Levels: 1, 2, 3

CHAPTER 11

# v11 HTTP security configuration verification requirements

## 11.1 Only defined HTTP Request methods are accepted

Verify that the application accepts only a defined set of required HTTP request methods, such as GET and POST are accepted, and unused methods (e.g. TRACE, PUT, and DELETE) are explicitly blocked.

Levels: 1, 2, 3

## 11.2 Every HTTP Response contains a Content-Type header with safe character set

Verify that every HTTP response contains a content type header specifying a safe character set (e.g., UTF-8, ISO 8859-1).

Levels: 1, 2, 3

### General

Arshan Dabirsiaghi (see links) discovered that many web application frameworks allowed well chosen and/or arbitrary HTTP methods to bypass an environment level access control check: Many frameworks and languages treat "HEAD" as a "GET" request, albeit one without any body in the response. If a security constraint was set on "GET" requests such that only "authenticatedUsers" could access GET requests for a particular servlet or resource, it would be bypassed for the "HEAD" version. This allowed unauthorized blind submission of any privileged GET request Some frameworks allowed arbitrary HTTP methods such as "JEFF" or "CATS" to be used without limitation. These were treated as if a "GET" method was issued, and again were found not to be subject to method role based access control checks on a number of languages and frameworks, again allowing unauthorized blind submission of privileged GET requests. In many cases, code which explicitly checked for a "GET" or "POST" method would be safe.

- OWASP: Test HTTP Methods (OTG-CONFIG-006): Arbitrary HTTP Methods

Also note that custom methods may be used in:

- Denial Of Service attack (bypass reverse proxy cache to overload the server with expensive HTTP calls)
- Cache poisoning / defacement (if HEAD /, which does not return any content, is cached as GET /)

To test this use an HTTP testing tool like Curl to send a non-standard HTTP verb to the system.

Example of a system responding to a non-standard HTTP verb:

```
curl -v -X OWASP -I www.owasp.org

* About to connect() to www.owasp.org port 80 (#0)
*   Trying 192.237.166.62...
* connected
* Connected to www.owasp.org (192.237.166.62) port 80 (#0)
> OWASP / HTTP/1.1
> User-Agent: curl/7.24.0 (x86_64-apple-darwin12.0) libcurl/7.24.0 OpenSSL/0.9.8y␣
↪zlib/1.2.5
> Host: www.owasp.org
> Accept: */*
>
< HTTP/1.1 301 Moved Permanently
< Date: Thu, 05 Dec 2013 12:03:11 GMT
< Server: Apache
< Location: https://www.owasp.org/
< Vary: Accept-Encoding
< Content-Length: 230
< Content-Type: text/html; charset=iso-8859-1
<
* Connection #0 to host www.owasp.org left intact
* Closing connection #0
```

Example of a system responding the the appropriate error code:

```
curl -v -X OWASP -I google.com
* About to connect() to google.com port 80 (#0)
*   Trying 173.194.65.102...
* connected
* Connected to google.com (173.194.65.102) port 80 (#0)
> OWASP / HTTP/1.1
> User-Agent: curl/7.24.0 (x86_64-apple-darwin12.0) libcurl/7.24.0 OpenSSL/0.9.8y␣
↪zlib/1.2.5
> Host: google.com
> Accept: */*
>
< HTTP/1.1 405 Method Not Allowed
< Content-Type: text/html; charset=UTF-8
< Content-Length: 960
< Date: Thu, 05 Dec 2013 12:24:05 GMT
< Server: GFE/2.0
< Alternate-Protocol: 80:quic
<
* Connection #0 to host google.com left intact
* Closing connection #0
```

## 11.3 Trusted HTTP headers are authenticated

Verify that HTTP headers added by a trusted proxy or SSO devices, such as a bearer token, are authenticated by the application.

Levels: 2, 3

### General

See Rohit Raisinghani: How missing characterset can cause Security Vulnerability.

## 11.4 X-Frame-Options is used correctly

Verify that the Content Security Policy V2 (CSP) is in use for sites where content should not be viewed in a 3rd-party X-Frame.

Levels: 2, 3

## 11.5 X-Content-Type-Options is used correctly

Verify that the HTTP headers or any part of the HTTP response do not expose detailed version information of system components.

Levels: 1, 2, 3

## 11.6 HTTP headers in Requests and Responses contain only printable ASCII

Verify that all API responses contain X-Content-Type-Options: nosniff and Content-Disposition: attachment; filename="api.json" (or other appropriate filename for the content type).

Levels: 1, 2, 3

## 11.7 Content-Security-Policy is used correctly

Verify that the Content Security Policy V2 (CSP) is in use in a way that either disables inline JavaScript or provides an integrity check on inline JavaScript with CSP noncing or hashing.

Levels: 1, 2, 3

## 11.8 X-XSS-Protection is used correctly

Verify that the X-XSS-Protection: 1; mode=block header is in place.

Levels: 1, 2, 3

## General

Clickjacking, also known as a "UI redress attack", is when an attacker uses multiple transparent or opaque layers to trick a user into clicking on a button or link on another page when they were intending to click on the the top level page. Thus, the attacker is "hijacking" clicks meant for their page and routing them to other another page, most likely owned by another application, domain, or both. Using a similar technique, keystrokes can also be hijacked. With a carefully crafted combination of stylesheets, iframes, and text boxes, a user can be led to believe they are typing in the password to their email or bank account, but are instead typing into an invisible frame controlled by the attacker. * OWASP: Clickjacking

CHAPTER 12

# v12 Security configuration verification requirements

CHAPTER 13

# v13 Malicious controls verification requirements

## 13.1 No malicious code in custom code

Verify all malicious activity is adequately sandboxed, containerized or isolated to delay and deter attackers from attacking other applications.

Levels: 3

## 13.2 Code, libraries, executables and configuration files are verfied

Verify that a code review looks for malicious code, back doors, Easter eggs, and logic flaws.

Levels: 3

CHAPTER 14

# v14 Internal security verification requirements

# CHAPTER 15

# v15 Business logic verification requirements

## 15.1 Appropriately uses a trusted environment

Verify the application will only process business logic flows in sequential step order, with all steps being processed in realistic human time, and not process out of order, skipped steps, process steps from another user, or too quickly submitted transactions.

Levels: 2, 3

## 15.2 Does not allow spoofed high value transactions

Verify the application has business limits and correctly enforces on a per user basis, with configurable alerting and automated reactions to automated or unusual attack.

Levels: 2, 3

CHAPTER 16

# v16 Files and resources verification requirements

## 16.1 Safe from unsafe redirects

Verify that URL redirects and forwards only allow whitelisted destinations, or show a warning when redirecting to potentially untrusted content.

Levels: 1, 2, 3

### General

> Unvalidated redirects and forwards are possible when a web application accepts untrusted input that could cause the web application to redirect the request to a URL contained within untrusted input. By modifying untrusted URL input to a malicious site, an attacker may successfully launch a phishing scam and steal user credentials. Because the server name in the modified link is identical to the original site, phishing attempts may have a more trustworthy appearance. Unvalidated redirect and forward attacks can also be used to maliciously craft a URL that would pass the application's access control check and then forward the attacker to privileged functions that they would normally not be able to access. * OWASP: Unvalidated Redirects and Forwards Cheat Sheet

## 16.2 Safe from path traversal

Verify that untrusted file data submitted to the application is not used directly with file I/O commands, particularly to protect against path traversal, local file include, file mime type, and OS command injection vulnerabilities.

Levels: 1, 2, 3

### General

> A Path Traversal attack aims to access files and directories that are stored outside the web root folder. By browsing the application, the attacker looks for absolute links to files stored on the web server. By

manipulating variables that reference files with "dot-dot-slash (../)" sequences and its variations, it may be possible to access arbitrary files and directories stored on file system, including application source code, configuration and critical system files, limited by system operational access control. The attacker uses "../" sequences to move up to root directory, thus permitting navigation through the file system. This attack can be executed with an external malicious code injected on the path, like the Resource Injection attack. To perform this attack it's not necessary to use a specific tool; attackers typically use a spider/crawler to detect all URLs available. This attack is also known as "dot-dot-slash", "directory traversal", "directory climbing" and "backtracking". * OWASP: Path Traversal

### PHP

Keep it real with realpath.

## 16.3 Anti-virus scanning

Verify that files obtained from untrusted sources are validated to be of expected type and scanned by antivirus scanners to prevent upload of known malicious content.

Levels: 1, 2, 3

## 16.4 Safe from local file inclusion attacks

Verify that untrusted data is not used within inclusion, class loader, or reflection capabilities to prevent remote/local file inclusion vulnerabilities.

Levels: 1, 2, 3

## 16.5 Safe from remote file inclusion attacks

Verify that untrusted data is not used within cross-domain resource sharing (CORS) to protect against arbitrary remote content.

Levels: 1, 2, 3

### General

parameters obtained from untrusted sources

What is an untrusted source from the applications point of view? Is the User Agent untrusted? Even in an authenticated back-end with proper CSRF protection? Is an external webservice untrusted? Is an internal webservice untrusted? Is the database untrusted? Trust is not binary but what is important is that often these types of decisions are made without regard for security at all. More than giving a PASS or a FAIL it is important to specify which services there are and their implicit trust level.

So if, in order to get the current temperature of the users location based on IP the application gives the IP to a service, waits for a reply (blocking) and then outputs whatever it get's back into the header of every page you may surmise that this application has a deep trust relation with this service and question that trust relation.

When in doubt FAIL, so the customer will look at this.

canonicalized

Canonicalization or C14n is important to guarantee correctness. Say an application allows for ?page=PAYMENT and ?page=payment in it's routing but only checks for 'payment' in adding the CSRF token, you may disable CSRF.

Think of: * path (/var/www/product/../../../etc/passwd may pass a simple verification of 'starts with /var/www') * XML / HTML * URLs * Unicode

## 16.6 Resource sharing

Verify that files obtained from untrusted sources are stored outside the webroot, with limited permissions, preferably with strong validation.

Levels: 2, 3

### General

> Are IFRAME elements used to load in content from external parties (for instance from advertisers)? Can they be subverted to load in a URL from an attacker? Validate URLs passed to XMLHttpRequest.open, current browsers allow these URLs to be cross domain and this behavior can lead to code injection by a remote attacker. Pay extra attention to absolute URLs. * OWASP: HTML5 Security Cheat Sheet: Cross Origin Resource Sharing

## 16.7 Untrusted files are stored outside the webroot

Verify that the web or application server is configured by default to deny access to remote resources or systems outside the web or application server.

Levels: 2, 3

## 16.8 Deny access to resources or systems outside web or app server

Verify the application code does not execute uploaded data obtained from untrusted sources.

Levels: 1, 2, 3

## 16.9 Does not execute uploaded data from untrusted sources

Do not use Flash, Active-X, Silverlight, NACL, client-side Java or other client side technologies not supported natively via W3C browser standards.

Levels: 1, 2, 3

CHAPTER 17

# v17 Mobile verification requirements

## 17.1 App verifies SSL certificates

Verify that ID values stored on the device and retrievable by other applications, such as the UDID or IMEI number are not used as authentication tokens.

Levels: 1, 2, 3

### General

Risks: * Mobile Internet is an insecure channel * Public Wifi hotspots are open unsecured networks * Hotstpots at Coffee Shops, Book Stores, Airports * Plenty of open source tools available to sniff from open wireless networks * Firesheep addon for Firefox makes it easier * Grabs your Social Media and other web passwords with one click * Face Sniffer app for Android is the Firesheep version for Mobile devices to sniff passwords from open wireless networks * It is possible to throw a fake GSM signal. Chris Paget demonstrated a fake GSM tower during DefCon 2010 that costed about $1500. It is called IMSI catcher. An attacker can throw up a fake ATT / T-Mobile signal a few feet away. Your phone would connect to his tower since it would have a stronger signal than the nearest cell phone tower. All data that is sent unencrypted can be read by the attacker. * **‘OWASP: Security and Privacy issues in iOS and Android Apps <>‘__**

## 17.2 App does not use UDID values as security controls

Verify that the mobile app does not store sensitive data onto potentially unencrypted shared resources on the device (e.g. SD card or shared folders).

Levels: 1, 2, 3

### General

What are Device Identifiers ? Think of them as similar to the VIN number of a vehicle. 1. UDID (Unique Device Identifier) - Apple Serial Number 2. IMEI ( International Mobile Equipment Identity) Number - Unique GSM number, applicable to Android and iOS as long as it is on a GSM phone Most Apps collect at least one of the device identifiers App owners collect them Third party ad-networks that display banner ads inside the apps collect them Device IDs are collected because they now uniquely identify every device and the behavior of its user

Besides this information being Personable Identifiable Information (which should be handled securely) it can also be spoofed so is not a reliable identifier.

## 17.3 App protects sensitive data

Verify that sensitive data is not stored unprotected on the device, even in system protected areas such as key chains.

Levels: 1, 2, 3

## 17.4 App does not use SQLite for sensitive data

Verify that secret keys, API tokens, or passwords are dynamically generated in mobile applications.

Levels: 2, 3

### General

Storing sensitive information (i.e. PII, Passwords etc) local to the phone or device. Sensitive Data Could Include Username / Passwords Device IDs PII , SSN, Health Information Application Configuration Credit card numbers Why not? • Phones can be lost or stolen • Trivial to recover data if device is: • "jailbroken" • Rooted or • Not password protected • In other cases partial or full recovery of data may be still possible if there is physical access to the device Types of files where sensitive data may be present on Android apps Database files - SQL Lite files, *.db files SQL Lite Browser or Command line SQL Lite can be used to view them Regular ASCII files, log files and Binary Files Text Editors and Hex Editors can be used to view them

- OWASP: Security and Privacy issues in iOS and Android Apps: 2. Insecure data storage

## 17.5 App does not have Secret credentials hard-coded in executable

Verify that the mobile app prevents leaking of sensitive information (for example, screenshots are saved of the current application as the application is backgrounded or writing sensitive information in console) .

Levels: 2, 3

## 17.6 App protects against auto-snapshot information leakage

Verify that the application is requesting minimal permissions for required functionality and resources.

Levels: 2, 3

## 17.7 App cannot be run on a jailbroken or rooted device

Verify that the application sensitive code is laid out unpredictably in memory (For example ASLR).

Levels: 1, 2, 3

## 17.8 App session timeout is of a reasonable value

Verify that there are anti-debugging techniques present that are sufficient enough to deter or delay likely attackers from injecting debuggers into the mobile app (For example GDB).

Levels: 3

## 17.9 App requires appropriate permissions and resources

Verify that the app does not export sensitive activities, intents, content providers etc., for other mobile apps on the same device to exploit.

Levels: 1, 2, 3

## 17.10 App crash log does not contain sensitive data

Verify that mutable structures have been used for sensitive strings such as account numbers and are overwritten when not used. (Mitigate damage from memory analysis attacks).

Levels: 3

## 17.11 App binary has been obfuscated

Verify that the app's exposed activities, intents, content providers etc. validate all inputs.

Levels: 1, 2, 3

# CHAPTER 18

# v18 Web services verification requirements

## 18.1 Web Service client and server use same encoding

Verify that the same encoding style is used between the client and the server.

Levels: 1, 2, 3

## 18.2 Web Service admin is limited to admins

Verify that access to administration and management functions within the Web Service Application is limited to web service administrators.

Levels: 1, 2, 3

## 18.3 XML or JSON schemas are used properly

Verify that XML or JSON schema is in place and verified before accepting input.

Levels: 1, 2, 3

## 18.4 Input is size limited

Verify that all input is limited to an appropriate size limit.

Levels: 1, 2, 3

## 18.5 SOAP services comply to WS-I Basic Profile

Verify that SOAP based web services are compliant with Web Services-Interoperability (WS-I) Basic Profile at minimum.

Levels: 1, 2, 3

## 18.6 Session based authentication and authorization is used

Verify the use of session-based authentication and authorization. Please refer to sections 2, 3 and 4 for further guidance. Avoid the use of static "API keys" and similar.

Levels: 1, 2, 3

## 18.7 REST service is not vulnerable to CSRF

Verify that the REST service is protected from Cross-Site Request Forgery.

Levels: 1, 2, 3

## 18.8 REST service verifies Content-Type

Verify the REST service explicitly check the incoming Content-Type to be the expected one, such as application/xml or application/json.

Levels: 2, 3

## 18.9 Message payload is signed

Verify that the message payload is signed to ensure reliable transport between client and service.

Levels: 2, 3

## 18.10 No alternative (insecure) access paths

Verify that alternative and less secure access paths do not exist.

Levels: 2, 3

# CHAPTER 19

# v19 Configuration

## 19.1 Components are stripped, up-to-date and securely configured

All components should be up to date with proper security configuration(s) and version(s). This should include removal of unneeded configurations and folders such as sample applications, platform documentation, and default or example users.

Levels: 1, 2, 3

## 19.2 Communication between components is encrypted

Communications between components, such as between the application server and the database server, should be encrypted, particularly when the components are in different containers or on different systems.

Levels: 2, 3

## 19.3 Communication between components is authenticated with least privileges

Communications between components, such as between the application server and the database server should be authenticated using an account with the least necessary privileges.

Levels: 2, 3

## 19.4 Deployments are adequately sandboxed, containerized or isolated

Verify application deployments are adequately sandboxed, containerized or isolated to delay and deter attackers from attacking other applications.

Levels: 2, 3

## 19.5 Deployment processes are secure

Verify that the application build and deployment processes are performed in a secure fashion.

Levels: 2, 3

## 19.6 Admins can verify integrity of configuration

Verify that authorised administrators have the capability to verify the integrity of all security-relevant configurations to ensure that they have not been tampered with.

Levels: 3

## 19.7 Application components are signed

Verify that all application components are signed.

Levels: 3

## 19.8 Thrid party components come from trusted repos

Verify that third party components come from trusted repositories.

Levels: 3

## 19.9 Security flags are enabled

Ensure that build processes for system level languages have all security flags enabled, such as ASLR, DEP, and security checks.

Levels: 3

CHAPTER 20

# Level 1: Opportunistic

CHAPTER 21

# Level 2: Standard

CHAPTER 22

# Level 3: Advanced