9. Vulnerable Components

Caused when systems release software using vulnerable dependencies or when vulnerabilities are discovered in dependencies after release.

DEFENCE:

1. Use OWASP Dependency Checker as part of a continuous integration pipeline and continue to run against production builds after deploy.
2. Have effective patching/library updating processes in place.

10. Unvalidated Redirects and Forwards

Occurs when a server redirects (or forwards) a user request based on untrusted input. A classic example is a login form with an unsanitized 'sendTo' query string parameter allowing a user journey to be hijacked.

DEFENCE:

1. Only redirect/forward to known safe locations either by configuring a whitelist (and referencing by key) or having strong destination sanitizing.

8. Sensitive Data Exposure

Failure to use appropriate cryptographic algorithms to protect data either in transit or at rest.
Home-grown, weak/known compromised algorithms or inappropriate use of strong algorithms are the most common cause.
Also consider client (e.g. browser) behaviour/ features like caching/saved form data.
Inappropriate HTTP header use can cause sensitive information to be captured/logged en-route.

DEFENCE:

1. Don't store sensitive data that you don't need to.
2. Ensure that strong, standard, context appropriate algorithms are used.

4. Insecure Direct Object References

Insecure Direct Object References occur when an application exposes (e.g. database) resource keys to the web application and fails to check ownership/ access rights to that resource at request time.

DEFENCE:

1. Check access for any direct object reference, on every request.
2. Eliminate direct object references by using per user or per session keys.

5. Security Misconfiguration

Occurs when (e.g.) default credentials or development features (non production services, information leakages) are configured in production deployments.

DEFENCE:

1. Ensure development teams fully understand the frameworks and libraries that they use and that they are accountable for production config.

3. XSS

Cross Site Scripting occurs when untrusted input is sent to the browser for evaluation without proper escaping. XSS attacks may be Stored, Reflected or Dom-based.

```
<script>alert(0);</script>
```

DEFENCE:

1. Properly escape all untrusted data based on the correct context of the output (e.g. body, attribute, js, css, url).
2. Sanitize input (not a standalone defence)
3. Use Content Security Policies (CSP)

Top 10

OWASP

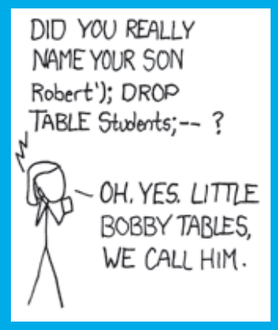
https://www.owasp.org

1. Injection

If untrusted input is sent to an interpreter (e.g. SQL database, LDAP store) then unintended commands may be executed causing confidentiality, integrity or availability issues.

DEFENCE:

1. Use libraries that avoid, or parameterise, the interpreter (e.g. prepared statements for SQL)
2. Sanitize input (not a standalone defence)



http://xkcd.com/327

2. Broken Auth

There are many ways that products can implement Auth incorrectly. Poor session management (fixation, insecure cookies, no timeout), weak policy (password strength, failure to lockout) and insecure handling of secrets (transmission and storage) are perhaps the most common.

DEFENCE:

1. Centralise authentication/session management to a single expert team
2. Provide simple interfaces for other products/ teams to use

6. CSRF

Cross-Site Request Forgery works when a victim visits a compromised site which then makes a request to the vulnerable domain.
Browsers pass cookies - even on cross-origin submissions - so any existing authentication will be hijacked.

(!) Neither CORS nor using specific HTTP verbs can protect against this.

DEFENCE:

1. Use a CSRF token
2. Synchronise cookie to header (for AJAX)
3. Ask for password on sensitive actions

7. Missing Access Control

Failure to correctly establish authorisation (e.g. relying on front-end hiding of features or obfuscation of remote services).

DEFENCE:

1. Ensure server/client boundaries are well understood and that all auth controls are implemented explicitly on server side.

EQUAL EXPERTS