OWASP

WEB SECURITY

INJECTION

**What is it?**

Allows attackers to access, modify and destroy data.

Attacker =====> malicious request ======> website =====> modified query =====> DB

<======= Data extraction <============== Query output <=============

**Example**

1. User accesses *http://mysite.com/widget?id=1*
2. This may transfer down to: *SELECT \* FROM widget WHERE id = 1;*
3. Attacker modifies the untrusted data

**Defences**

1. Whitelist untrusted data; e.g., validate. If you expect only an integer, check this and throw an exception if you receive anything else.
2. Parameterise SQL statements.
3. Fine tune DB permissions; e.g., only give access to the data that is needed. Lock down certain data.

BROKEN AUTHENTICATION

**What is it?**

An attacker can log onto a system and impersonate a user.

Authenticated request

Attacker ======================================================> Website

**\ /**

**\ /**

Attacker hijacking

**Example**

1. Authentication cookie theft - either through XSS attack, sniffing over an insecure connection (http), physically retrieving from users PC.
2. Session ID theft – copy and pasting a url with a session id, retrieving session id from logs accessible to the client.
3. Account management attack – brute force, weak credentials, exploit password reset (e.g., reset password and answer very basic security questions “mother’s maiden name”)

**Defences**

1. Protect cookies – use *httponly* flag and flag as *secure*
2. Decrease window of risk – expire sessions quickly. Retype old password when changing password.
3. Encourage strong passwords. log accounts on 3 login failures.

CROSS SITE SCRIPTING

**What is it?**

Attacker injects some client-side code in to a vulnerable field or URL. 2 types; persisted & reflected.

Attacker ===> URL with XSS payload ====> User ====> Website

<=== Client data exfiltrated <=== XSS payload “reflected”

**Example**

1. Attacker accesses vulnerable website: *https://mysite.com/search?q=lager*
2. This is a search site where the result is reflected back at the user; e.g. “you searched for lager”
3. q parameter is changed in the URL to some malicious JS code.
4. The URL is shared via twitter, email, etc. and users are exploited.

**Defence**

1. Whitelist untrusted data – similar to injection, validate everything a user can pass!
2. Encode output – never reflect untrusted data. Encode all output from DB (assume it is malicious)

INSECURE DIRECT OBJECT REFERENCES

**What is it?**

An attacker can have access to data not meant for them by exploiting a vulnerable website, e.g. by changing the integer ID in a url to get data from another user.

**Example**

1. Attacker logs in to website: *https://mybank.com/balance?accountId=293843*
2. The untrusted data is changed to another integer to access another users account

**Defences**

1. Implement access controls – be explicit about who can access resources.
2. Use indirect maps – e.g., don’t expose internal keys externally. If the integer in the url is 12345 – this could map to id 3 in the DB for example. Even better, map them to temporary ones which only persist for a user’s session.
3. Avoid predictable keys – e.g., incrementing integers

SECURITY MISCONFIGURATION

**What is it?**

A general term which can include a range of things:

* Out of date software
* Unnecessary features enabled or installed, e.g. open ports for services not used.
* Default accounts and their passwords still unchanged (e.g. admin/password for router access).
* Errors display stack / internals on the client side in production/

**Defences**

1. Turn off features that are not needed.
2. Be conscious of 3rd party tool risks – have a strategy to monitor and update
3. Ensure the app is production ready – server config, api key storage, log levels.

SENSITIVE DATA EXPOSURE

**What is it?**

Another general term which can include a range of things:

* Sniffing requests over http for passwords
* Revealing sensitive user data in logs
* Unnecessarily storing sensitive data which are vulnerable to be retrieved via other attacks
* Insufficient use of SSL – cookies not set securely, login not loaded over https
* Bad cryptography – incorrect password storage, poor protection of keys, weak algorithms chosen.
* Other – disclosure via URL (e.g., credit card details displayed in url), leaked info via logs.

**Defences**

1. You can’t lose what you don’t have. Think about whether you really need to store data, such as credit card information.
2. Apply https everywhere.
3. Use strong cryptography - use proven hashing algorithms for passwords etc.

CROSS SITE REQUEST FORGERY

**What is it?**

The attacker tricks the users browser into making authenticated requests on the users behalf.

Attacker ===> share malicious link ====> user ====> attackers website

<==== Malicious request

======> Target website

Authenticated request <=====

**Example**

1. The user finds out how a certain vulnerable site makes a certain request, for example, a bank transfers data via a http POST to a certain url with a certain request payload, and how this request is authenticated, e.g. via a auth cookie.
2. The attacker tricks the user in to clicking on their website, which sends that POST request with the relevant information (the attacker is assuming the user is logged in, and if so, the request will be successful)

**Defences**

1. Use anti-forgery tokens – hidden data on the form which only the client knows. This token is checked during requests.
2. Validate the referrer – only allow external requests from specific domains.
3. Native browser defences – cors. Do not rely on this as this is only specific to browsers.